Generative Models

Diane Lingrand and Frédéric Precioso

UNIVERSITÉ
CÔTE D'AZUR
Master Data Science DSAI M1

2021 - 2022

Outline

Introduction

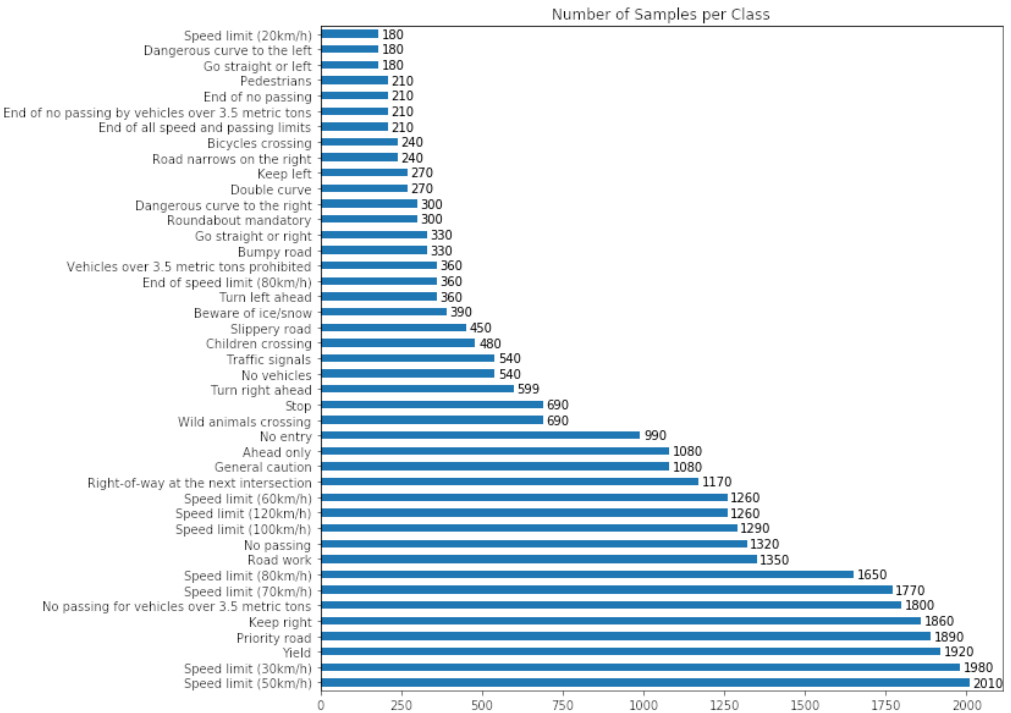
https://this-person-does-not-exist.com/en

Data augmentation (1)

- When you do not have enough data, you can artificially augment data with adapted transformations
  - For images, you can flip horizontally and/or vertically the images, crop them, change illumination, add some noise, and so on
  - For text, you can replace words by synonyms
  - For any other type of signal, you can always try to add noise but it has to preserve signal structure ... which is not always easy.

Data augmentation (2)

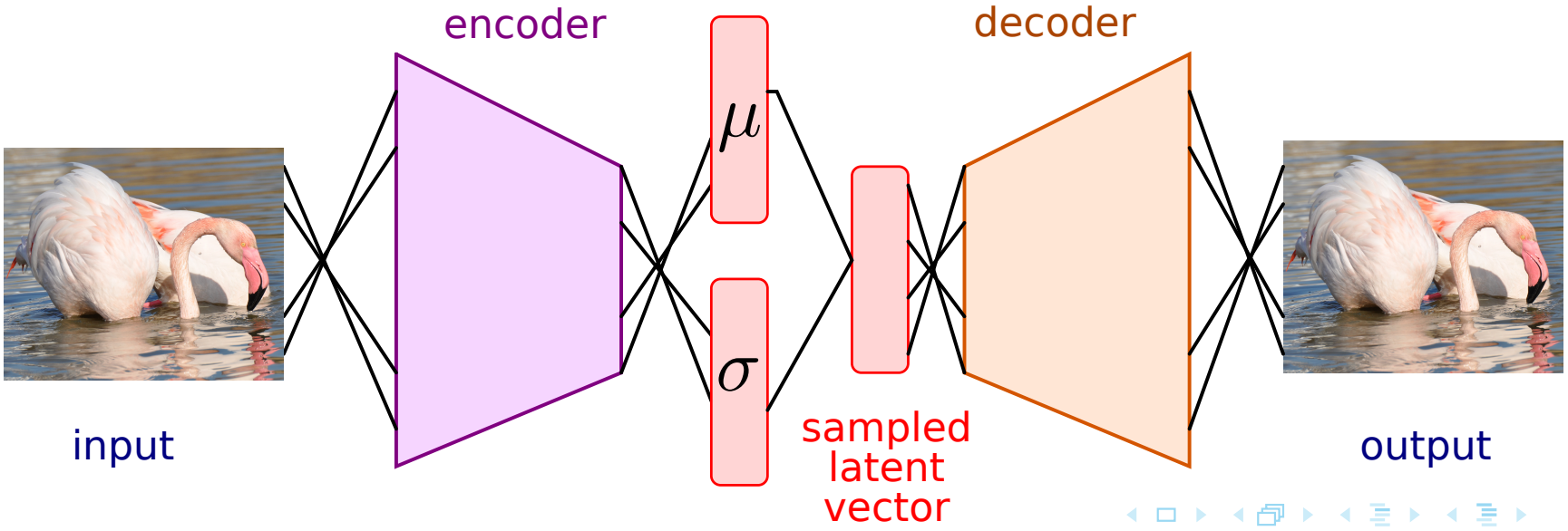
- It can also be interesting to apply data augmentation to balance representativeness between classes.



Outline

Variational Autoencoder (VAE)

- Autocoders learn a latent representation of data
- Variational autoencoders learn the parameter of a probability function modeling the latent data
  - ex: latent parameters μ and σ, assuming the distribution is gaussian
- It is thus a generative model able to generate similar data
  - regularisation of distribution parameters (mean and variance) in order to ensure that:
    - 2 samples similar in the latent space should have similar content when decoded
    - a data sampled from the latent space should generate meaningful content when decoded



Reparameterization trick

- problem with sampling: how to compute the gradient ?
- trick: consider variable z:

$$z = \mu + \sigma \odot \epsilon$$

  where $\epsilon \sim \mathcal{N}(0, 1)$
- learn μ and σ
- z is generated from μ, σ and ϵ
- backpropagation of gradients through μ and σ

VAE in practice: example using MNIST

VAE in practice: encoder

```
# this is an example of architecture.
# Feel free to add other layers
dimLatent = 28
inputImage = Input(shape=(784,))
encoded = Dense(256, activation='relu')(inputImage)
zMu = Dense(dimLatent)(encoded)
zLogSigma = Dense(dimLatent)(encoded)
```

VAE in practice: sample generation

```
from tensorflow.keras import backend as bk

def sampling(args):
    mu, logSigma = args
    epsilon = bk.random_normal(shape=(bk.shape(mu)[0], dimLatent), m
    # z = mu + sigma * epsilon
    return mu + bk.exp(logSigma) * epsilon

# Lambda layer for wrapping the sampling function
z = Lambda(sampling)([zMu, zLogSigma])

#encoder model
encoder = Model(inputImage, [zMu, zLogSigma, z])
```

VAE in practice: decoder and whole model

```
# symmetric to the encoder
latent = Input(shape=(dimLatent,))
decoded = Dense(256, activation='relu')(latent)
decoded = Dense(size1, activation='sigmoid')(decoded)
decoder = Model(latent, decoded)

# whole model to be trained
# z in the element at index 2 from [zMu, zLogSigma, z]
outputImage = decoder(encoder(inputImage)[2])
vae = Model(inputImage, outputImage)
```

VAE in practice: loss function

```
from tensorflow.keras.losses import mean_squared_error
## Loss function is composed of 2 terms:
# 1. reconstruction loss
reconstruction_loss = mean_squared_error(inputImage, outputImage)
reconstruction_loss *= size1
print(reconstruction_loss)
# 2. Kullback-Leibner loss
kl_loss = 1 + zLogSigma - bk.square(zMu) - bk.exp(zLogSigma)
kl_loss = bk.sum(kl_loss, axis=-1)
kl_loss *= -0.5
print(zMu)
print(kl_loss)
# vae loss
vae_loss = bk.mean(reconstruction_loss + kl_loss)
vae.add_loss(vae_loss)
```

Examples using MNIST

- original test images and decodings. Latent dimension = 28

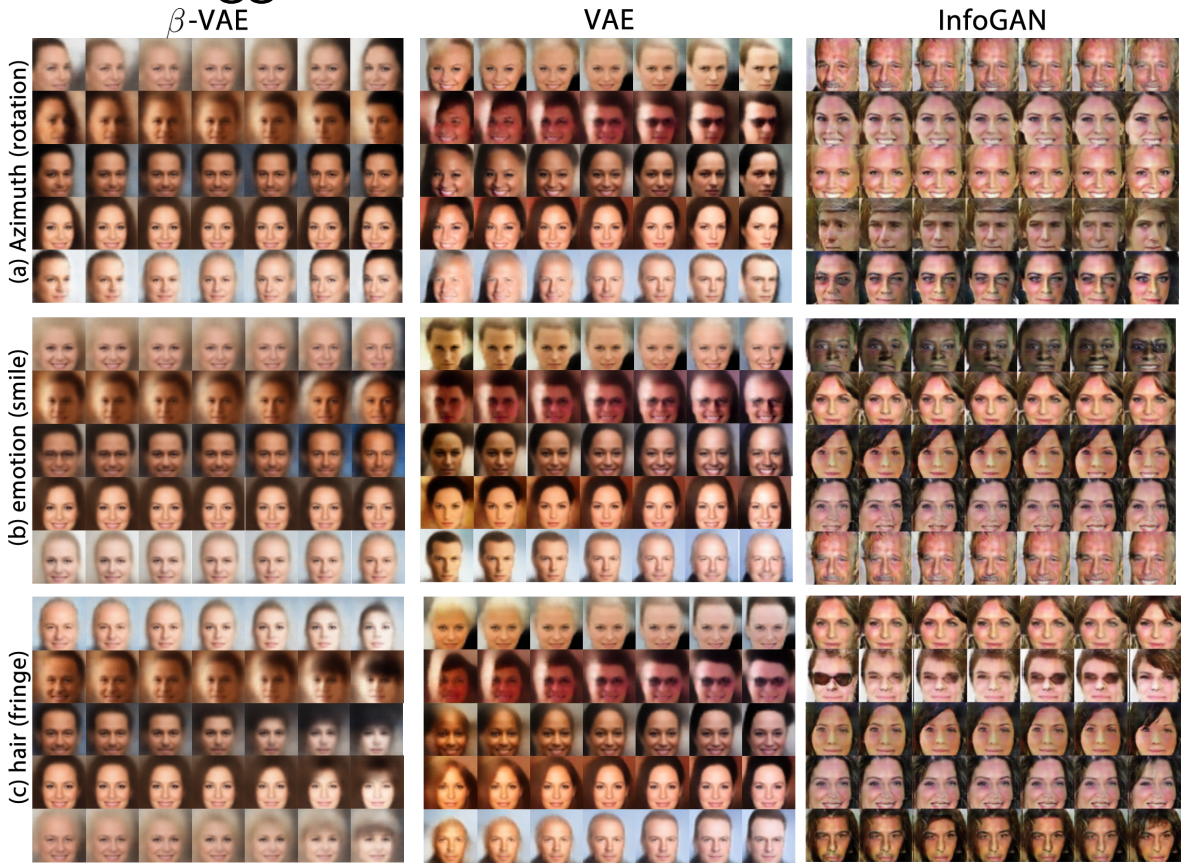
- from a test image, generate several latent representation and decodings

- interpolations between latent representation of 2 digits and decodings

- random latent representations and decodings

Disentangled Variational Autoencoder (β-VAE)

- ensure no correlation between latent variables
- param. β: weight to Kullback-Leibler divergence in the loss fn
- From Higgins etal, ICLR 2017 `https://openreview.net/pdf?id=Sy2fzU9gl`



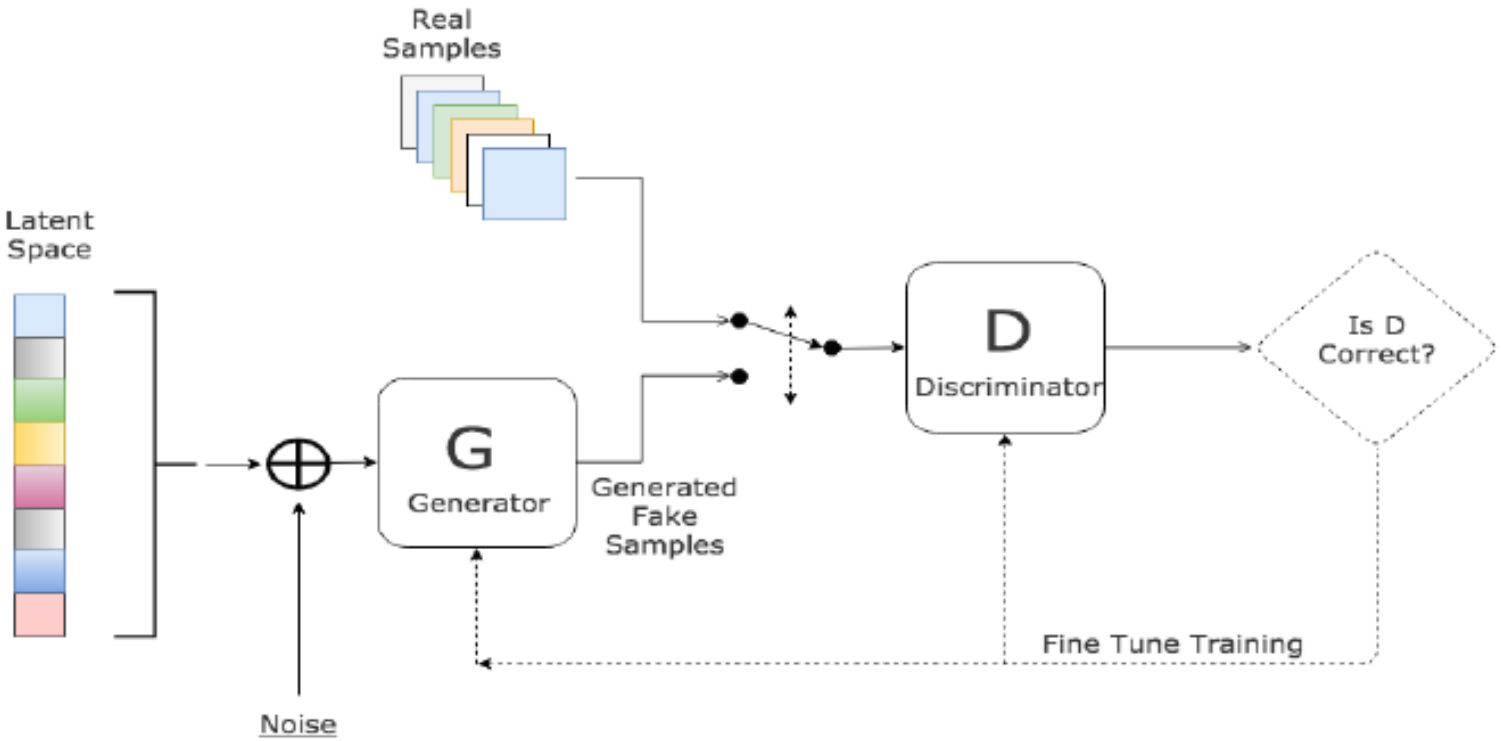
- implementation:

`https://github.com/Knight13/beta-VAE-disentanglement` or

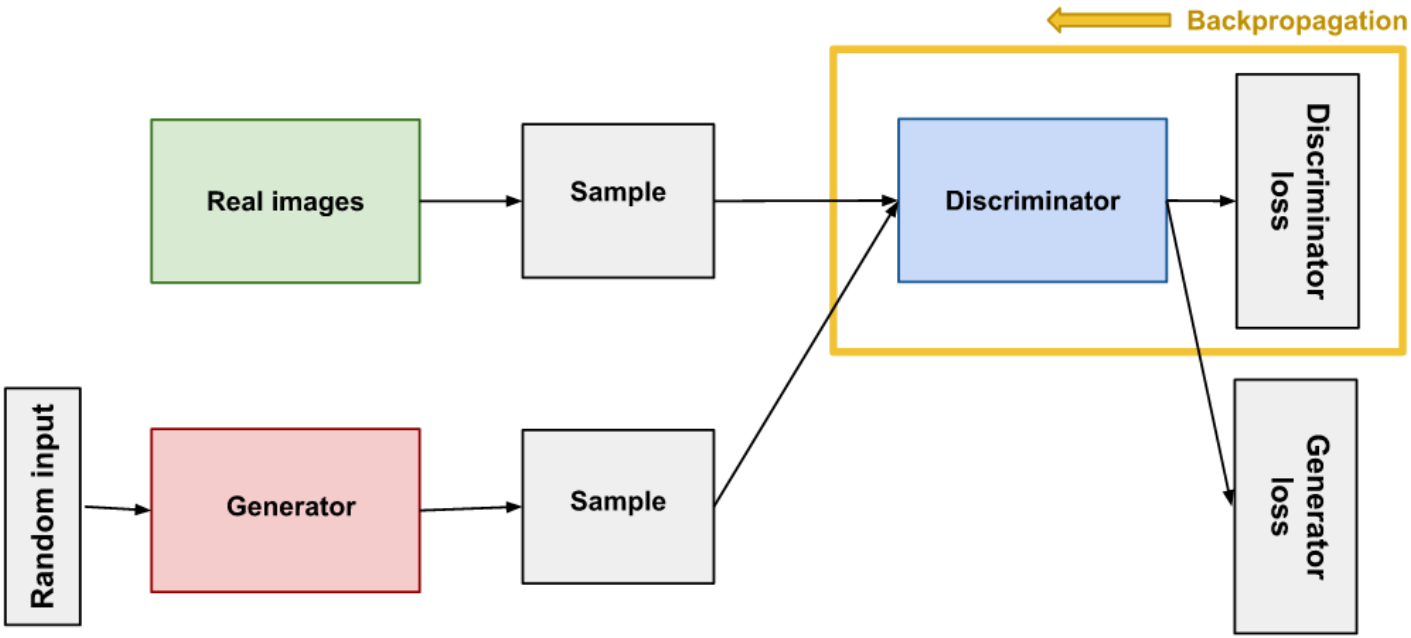
`https://blog.keras.io/building-autoencoders-in-keras.html`

Outline

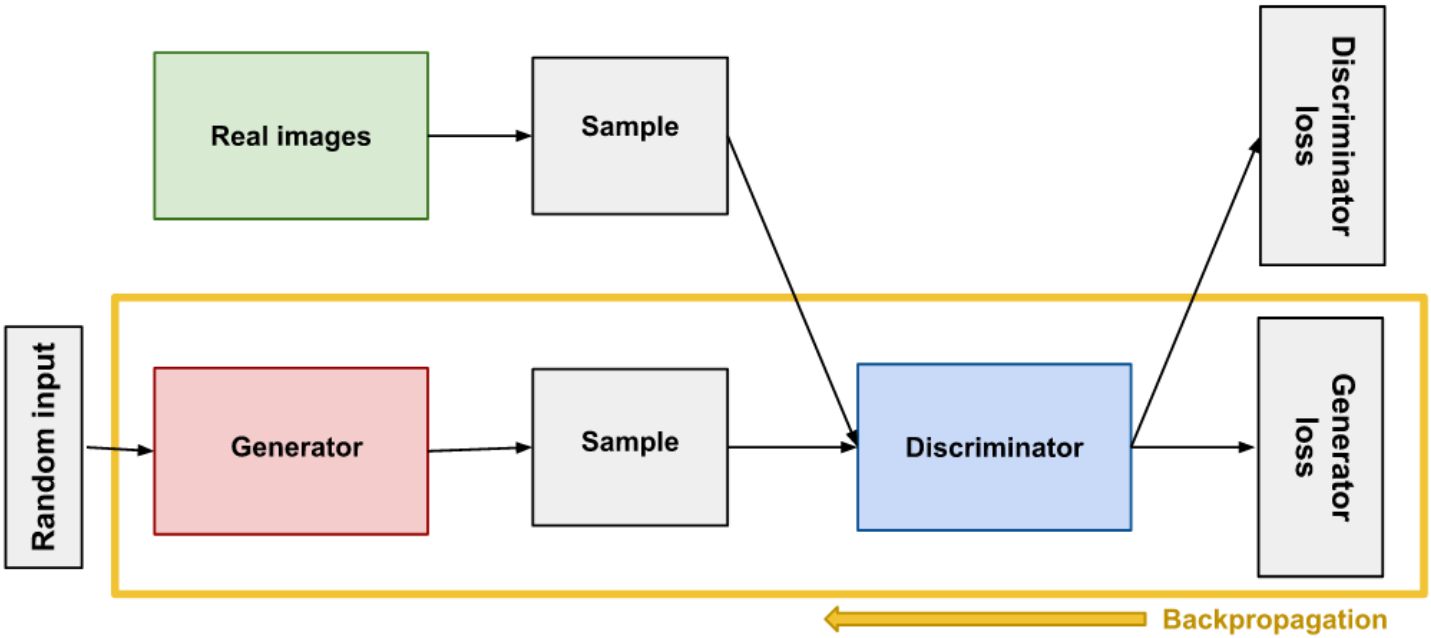
Generative Adversarial Networks



Training the Discriminator



Training the Generator



Algorithm

From `https://arxiv.org/pdf/1406.2661.pdf`

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Loss functions

- Minimax loss:

$$E_x[\log(D(x))] + E_z[\log(1 - D(G(z)))]$$

  - min for the generator
  - max for the discriminator
- Modified minimax: the generator maximizes $\log(D(G(z)))$
- Wasserstein loss (Earth Mover Distance between true and fake data)
  - D Loss: $D(x) - D(G(z))$
  - G Loss: $D(G(z))$

Try to play with GANs ?

Tensorflow has implemented the GANs and a tutorial is available.
See: `https://colab.research.google.com/github/tensorflow/gan/blob/master/tensorflow_gan/examples/colab_notebooks/tfgan_tutorial.ipynb?utm_source=ss-gan&utm_campaign=colab-external&utm_medium=referral&utm_content=tfgan-intro`

Problems

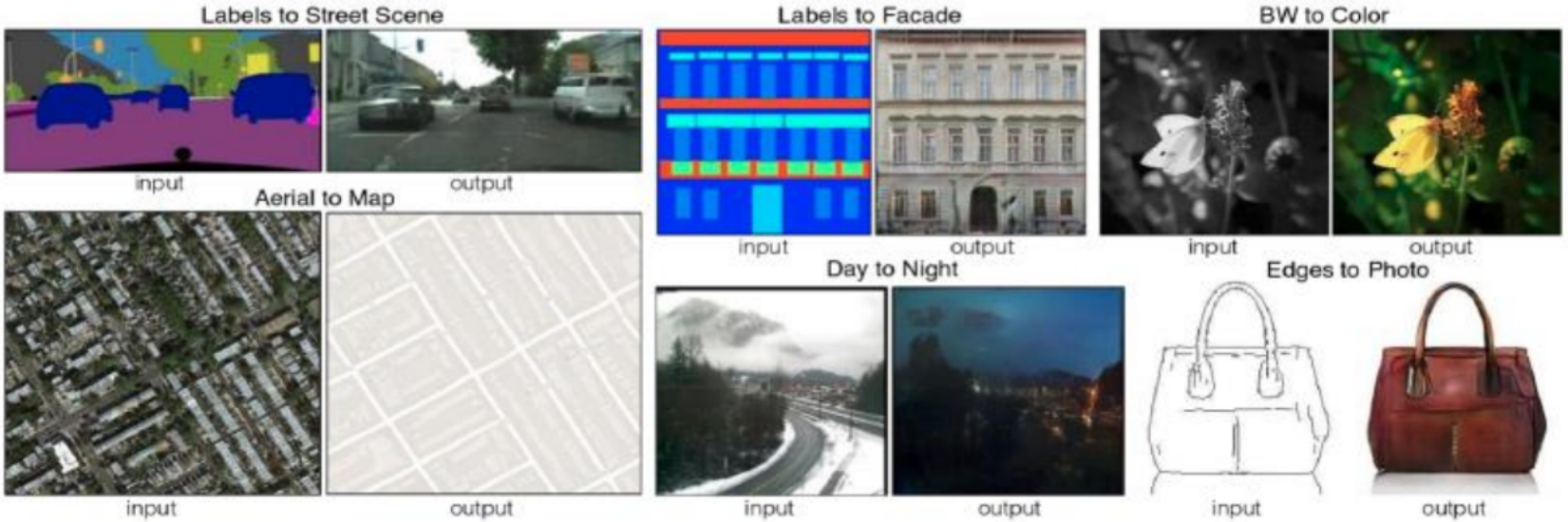
- Vanishing gradient
  - if the discriminator is too good
  - need to modify the loss function (Wasserstein or modified minimax)
- Mode collapse
  - always the same data generated (or a few number of)
  - Wasserstein loss or Unrolled GANs (where the generator cannot optimize for a single discriminator)
- Failure to convergence
  - really a problem
  - regularisations

GANs ...

- ... failed at finding adversarial data but ...
- are really usefull for other tasks !

(image from Karras et al. 2019)

GANs for other tasks



Three categories of GANs

- Typical GAN

$$\begin{bmatrix} -0.3 \\ 0.1 \\ \vdots \\ 0.9 \end{bmatrix}$$

random vector → Generator → 

image

- Conditional GAN

blue eyes,
red hair,
short hair

paired data

"Girl with red hair"
text → Generator → 

image

- Unsupervised Conditional GAN

domain x

domain y

unpaired data

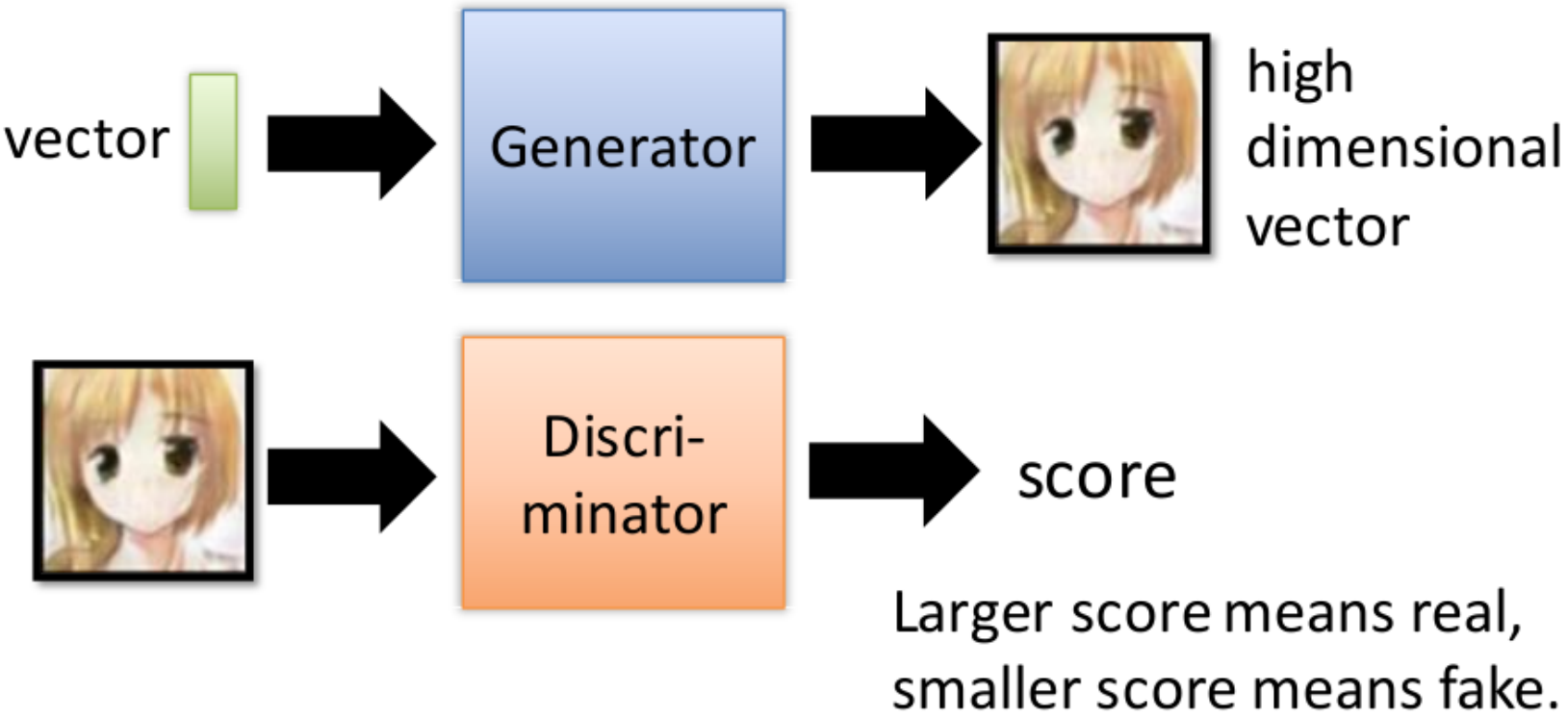
x

Photo → Generator → y

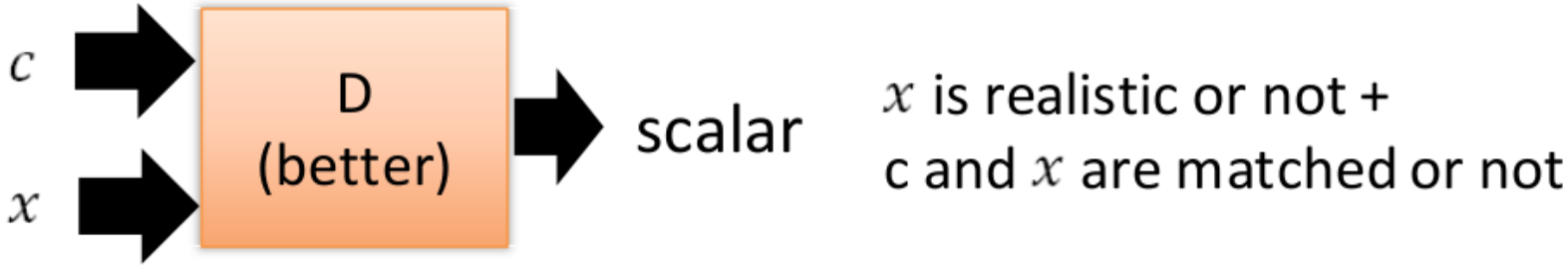
Vincent van Gogh's style

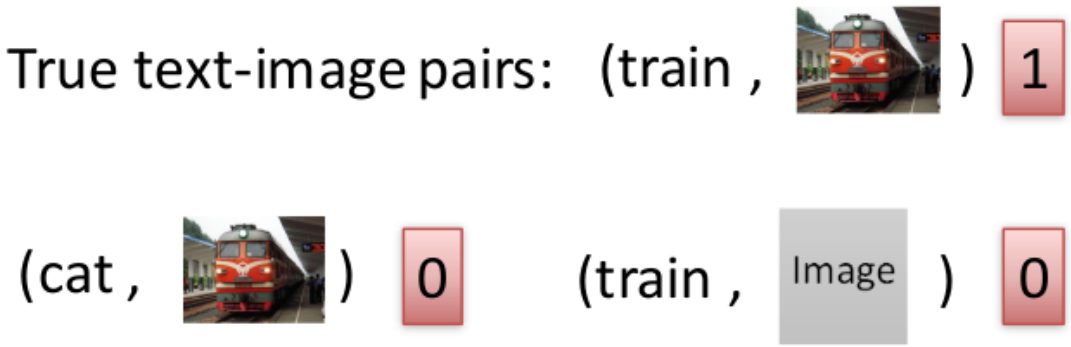
GAN example

- Anime face generation as example



Text-to-Image





[Scott Reed, et al, ICML, 2016]

Image-to-Label

Multi-label Image Classifier

person, sports ball,
baseball bat, baseball glove

Input condition

Generated output