

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ



Adversarial Attacks, GANs, and VAEs

Mohammad Taher Pilehvar

Machine Learning 01



TEIAS | Tehran Institute for
Advanced Studies

<https://teias-courses.github.io/ml01>

Some of the slides were borrowed from Stanford's CS230

Today

- Adversarial attacks
 - FGSM: The Fast Gradient Signed method
 - One pixel attack
 - Adversarial patch
- Adversarial defence
- Generative Adversarial Networks (GANs)
- Variational Autoencoders (VAEs)

Adversarial Attacks

<https://arxiv.org/abs/1904.08653>

Adversarial examples

- Specialized inputs created with the purpose of confusing a neural network

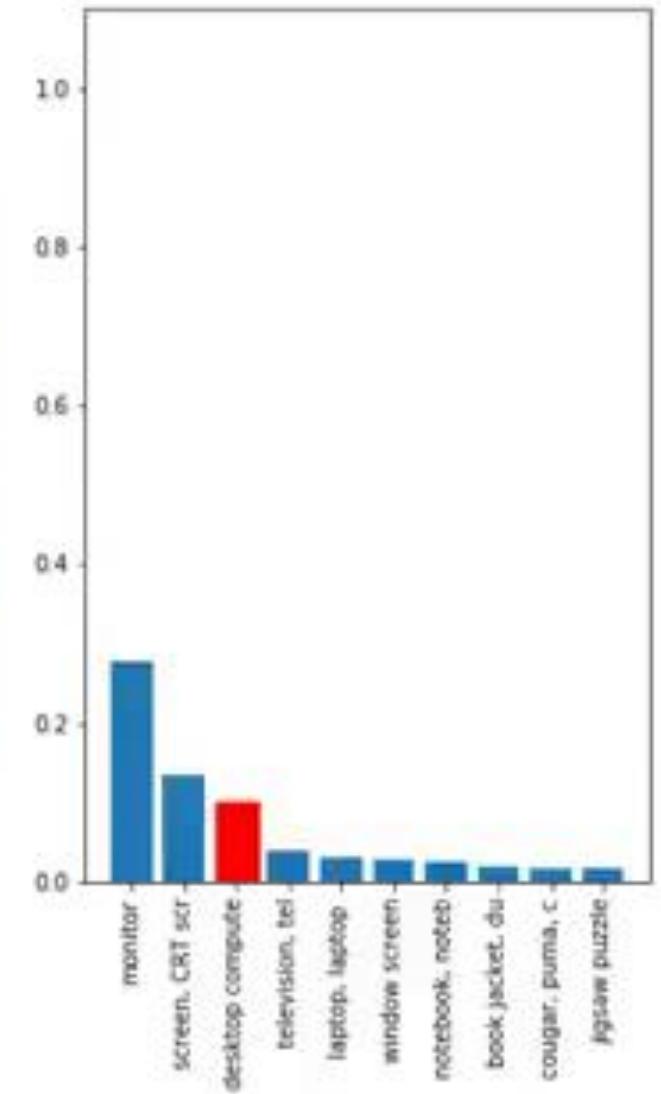


Adversarial Attacks

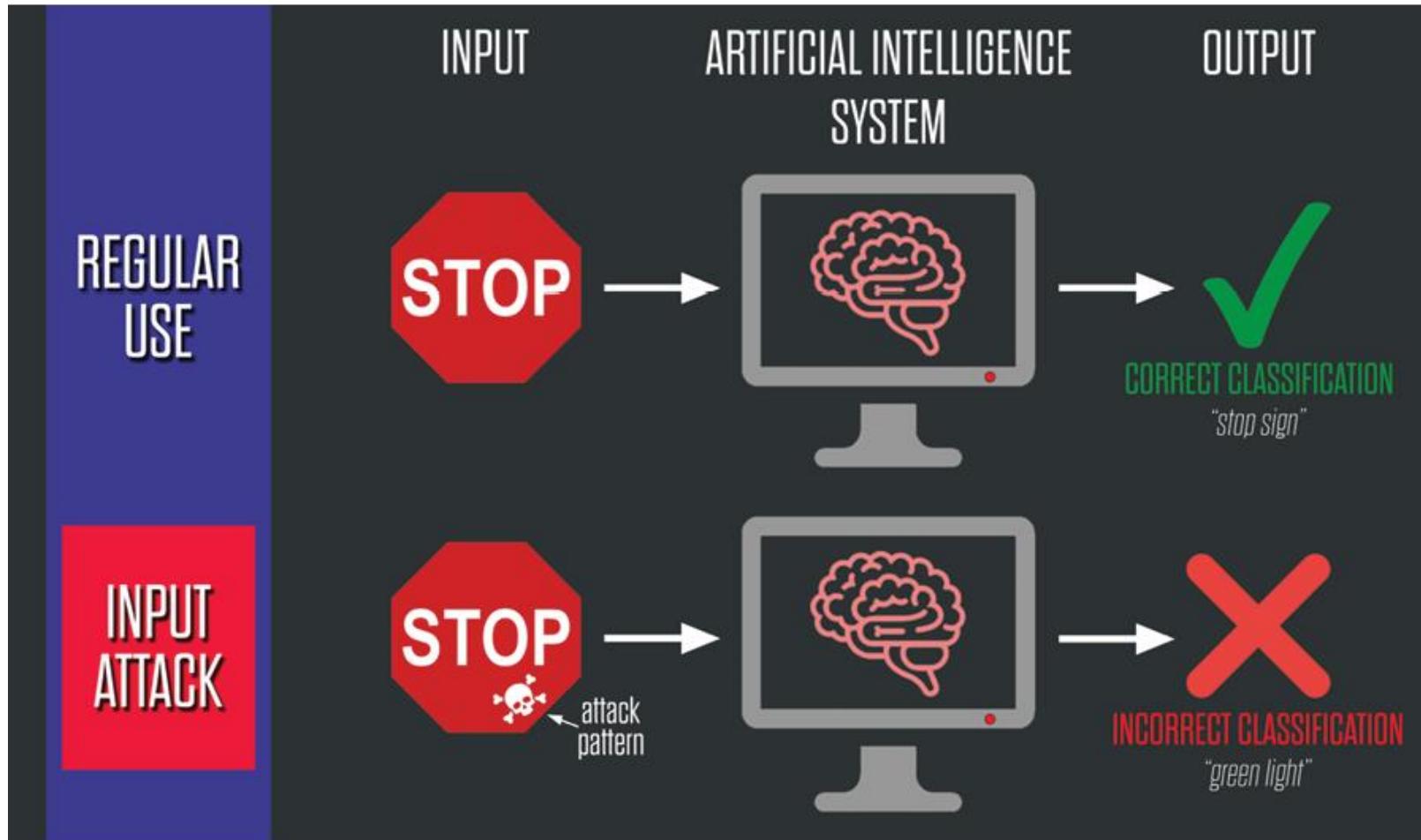
OpenAI

Adversarial examples

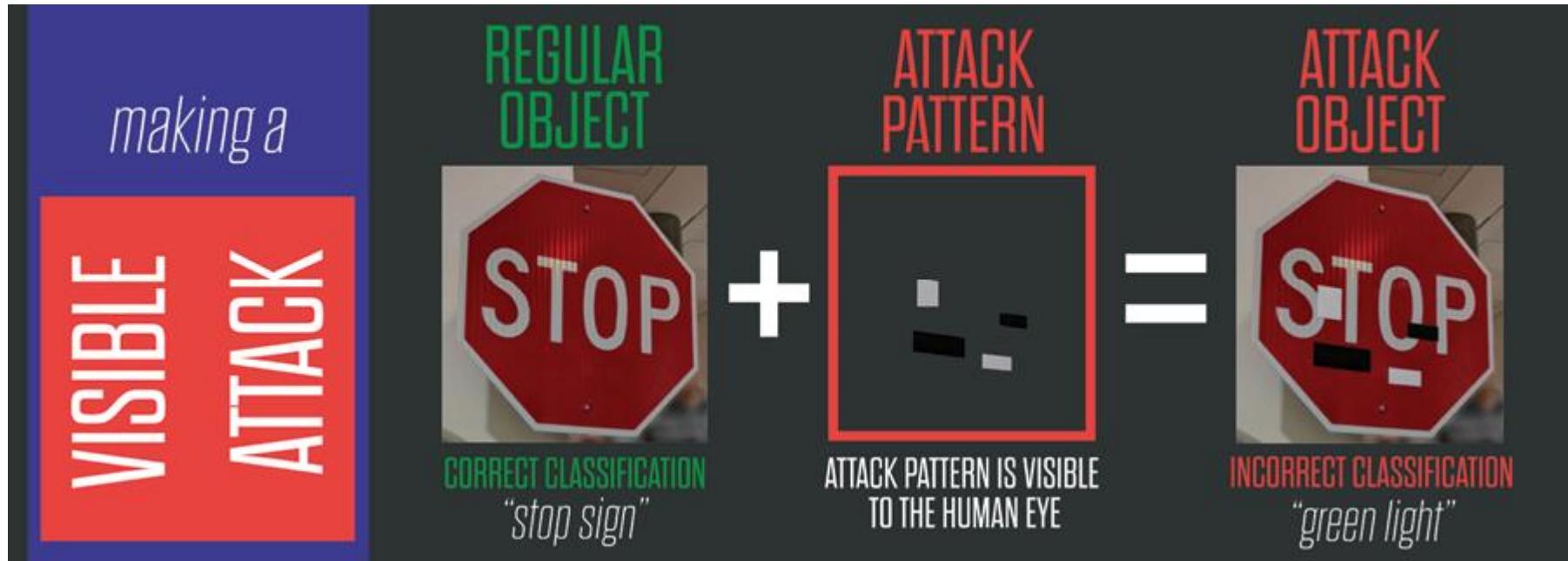
- Specialized inputs created with the purpose of confusing a neural network



Adversarial Attacks



Adversarial Attacks



Read more:

<https://www.belfercenter.org/publication/AttackingAI>

Adversarial Attacks

Adversarial examples

- Specialized inputs created with the purpose of confusing a neural network
- **White box attack**
 - Complete access to the model being attacked
- **Black box attack**
 - No knowledge of model internal architectures or training parameters

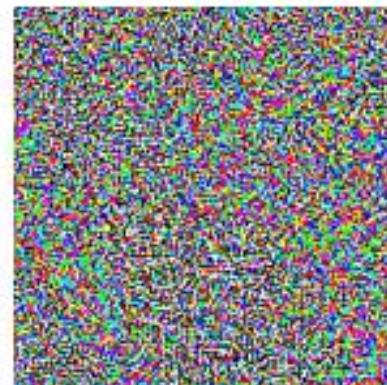
Adversarial Attacks

- White box attack
 - FGSM: Fast Gradient Signed Method (Goodfellow et al, 2014)



\mathbf{x}
“panda”
57.7% confidence

+ .007 ×



$\text{sign}(\nabla_{\mathbf{x}} J(\theta, \mathbf{x}, y))$
“nematode”
8.2% confidence

=



$\mathbf{x} +$
 $\epsilon \text{sign}(\nabla_{\mathbf{x}} J(\theta, \mathbf{x}, y))$
“gibbon”
99.3 % confidence

Adversarial Attacks - FGSM

Create an image that maximizes the loss

$$adv_x = x + \epsilon * \text{sign}(\nabla_x J(\theta, x, y))$$

- adv_x : Adversarial image.
- x : Original input image.
- y : Original input label.
- ϵ : Multiplier to ensure the perturbations are small.
- θ : Model parameters.
- J : Loss.



FGSM – other variants

- A) **Target Class Method:** This variant of FGSM [43] approach maximizes the probability of some specific target class y_{target} , which is unlikely the true class for a given example. The adversarial example is crafted using the following equation:

$$X_* = X - \epsilon * sign(\nabla_x J(X, y_{target}))$$

- B) **Basic Iterative Method:** This is a straightforward extension of the basic FGSM method [43]. This method generates adversarial samples iteratively using small step size.

$$X_*^0 = X; \quad X_*^{n+1} = Clip_{X,e}\{X_*^n + \alpha * sign(\nabla_x J(X_*^n, y_{true}))\}$$

Here, α is the step size and $Clip_{X,e}\{A\}$ denotes the element-wise clipping of X . The range of $A_{i,j}$ after clipping belongs in the interval $[X_{i,j} - \epsilon, X_{i,j} + \epsilon]$. This method does not typically rely on any approximation of the model and produces additional harmful adversarial examples when run for more iterations.

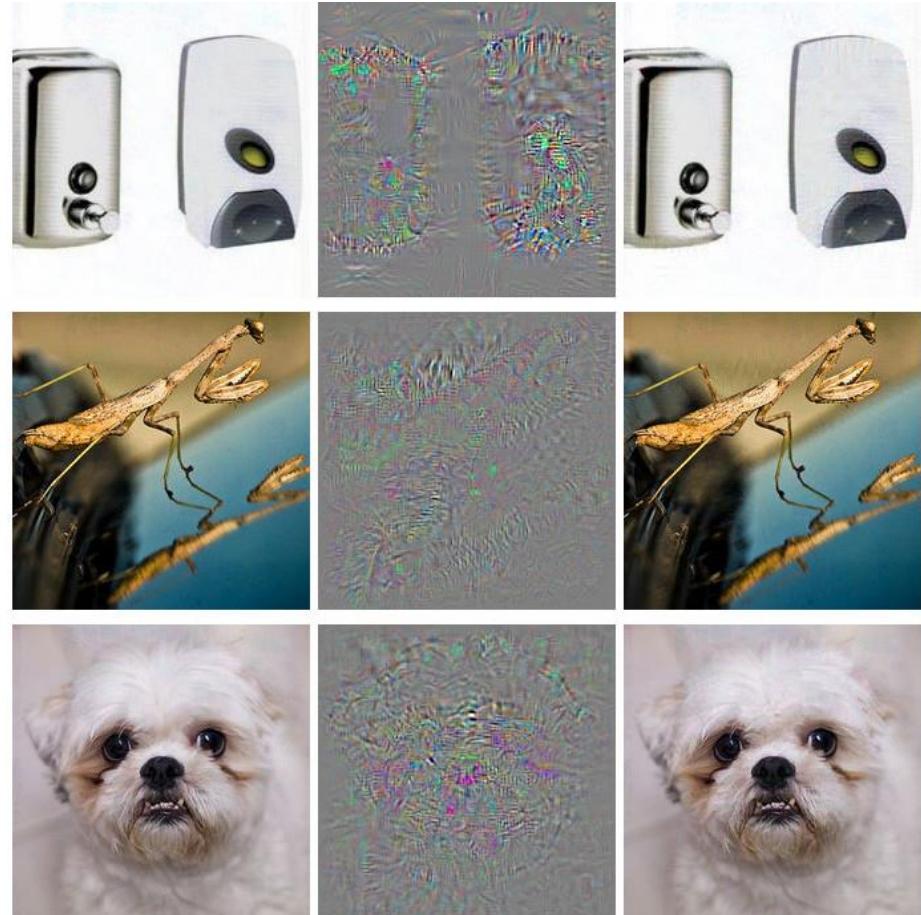
Adversarial Attacks

- Szegedy et. al (2013)

Minimize the following function w.r.t r

$$\text{loss}(\hat{f}(x + r), l) + c \cdot |r|$$

- x is the input
- r is the change to the input
- l is the desired outcome class
- c adjusts the distance between images and the distance between predictions



All images in the right column are predicted to be an ostrich

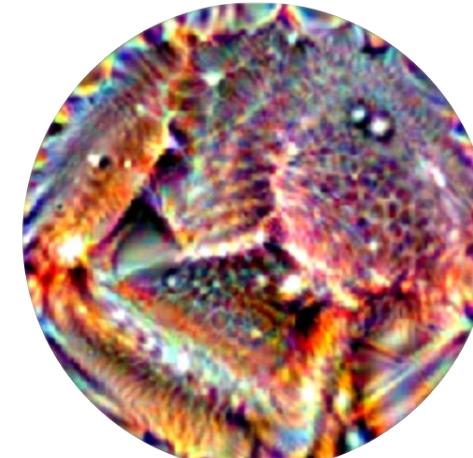
Adversarial Patch

Brown et al (2017)

<https://arxiv.org/abs/1712.09665>



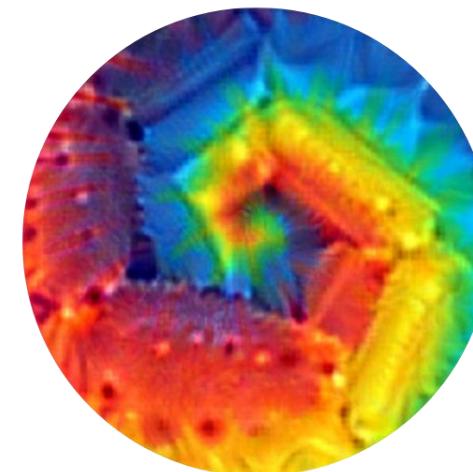
Banana



Crab

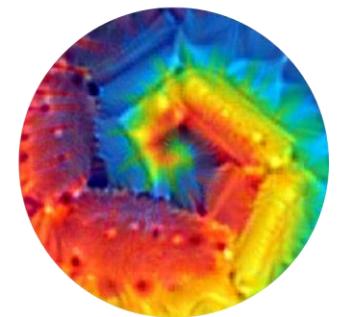
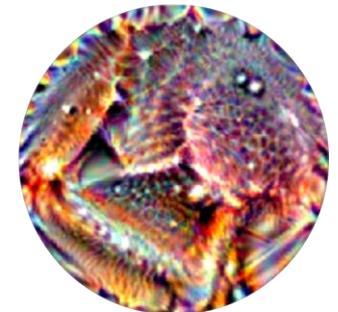
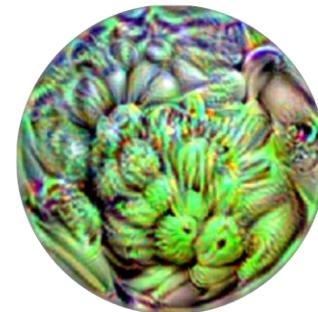
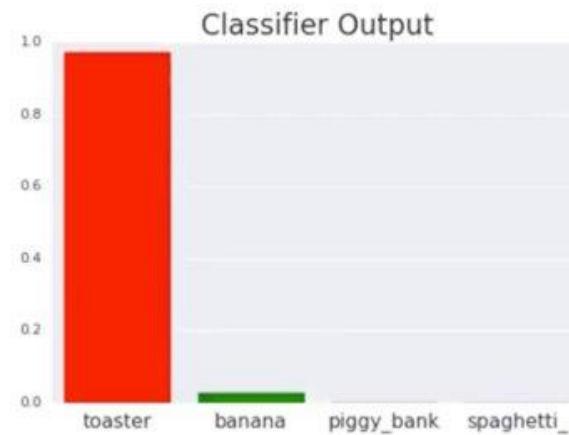
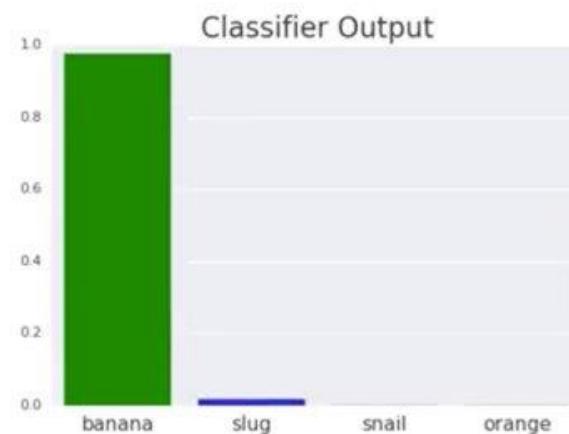
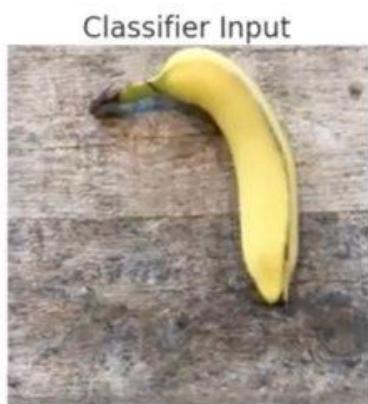


Toaster



Crab (disguised)

Adversarial Patch



One Pixel Attack

Su et al (2019)

- <https://arxiv.org/abs/1710.08864>



Cup(16.48%)
Soup Bowl(16.74%)



Bassinet(16.59%)
Paper Towel(16.21%)



Teapot(24.99%)
Joystick(37.39%)



Hamster(35.79%)
Nipple(42.36%)

Adversarial Defense

- Adversarial training
 - Brute force
 - Train the model on many (automatically generated) adversarial examples not to be fooled by them
 - [CleverHans](#): open-source implementation
- Defensive distillation
 - Papernot et al (2016): <https://arxiv.org/abs/1511.04508>
 - Attack methods usually rely on large gradients
 - Distilling a network reduces gradients

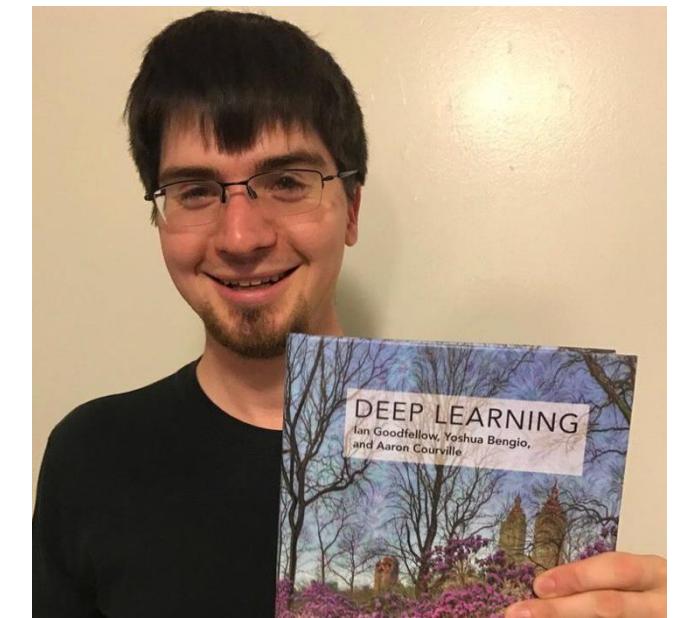
Read more

Adversarial Attacks and Defenses in Deep Learning

- Ren et al (2020)

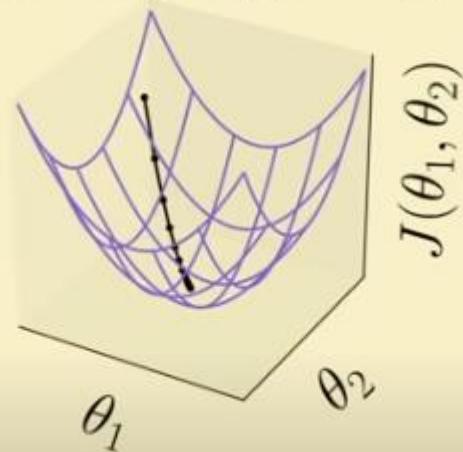
Generative Adversarial Networks (GANs)

Introduced in 2014 by Ian Goodfellow et al

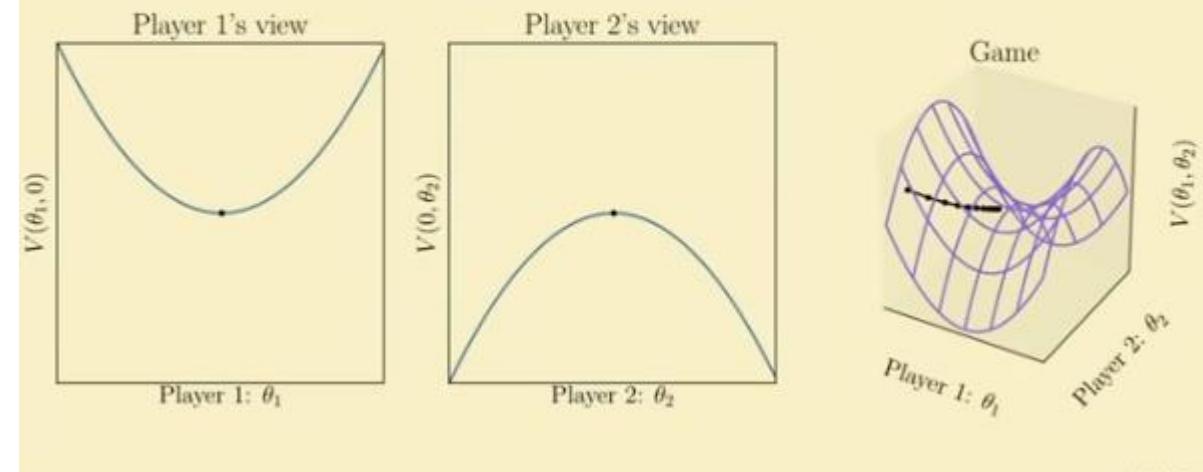


Adversarial ML

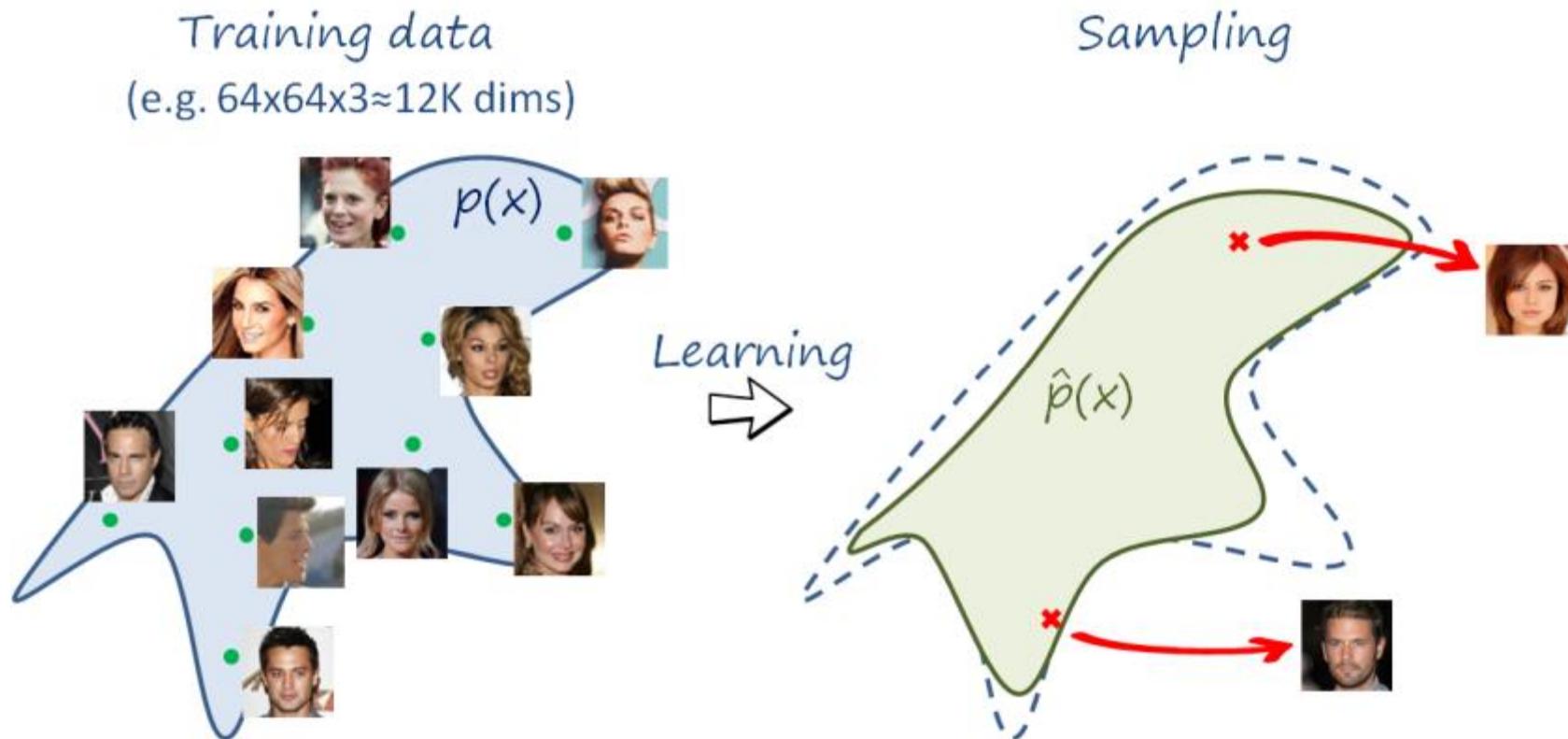
Most Traditional Machine Learning: Optimization



Adversarial Machine Learning:
Game Theory

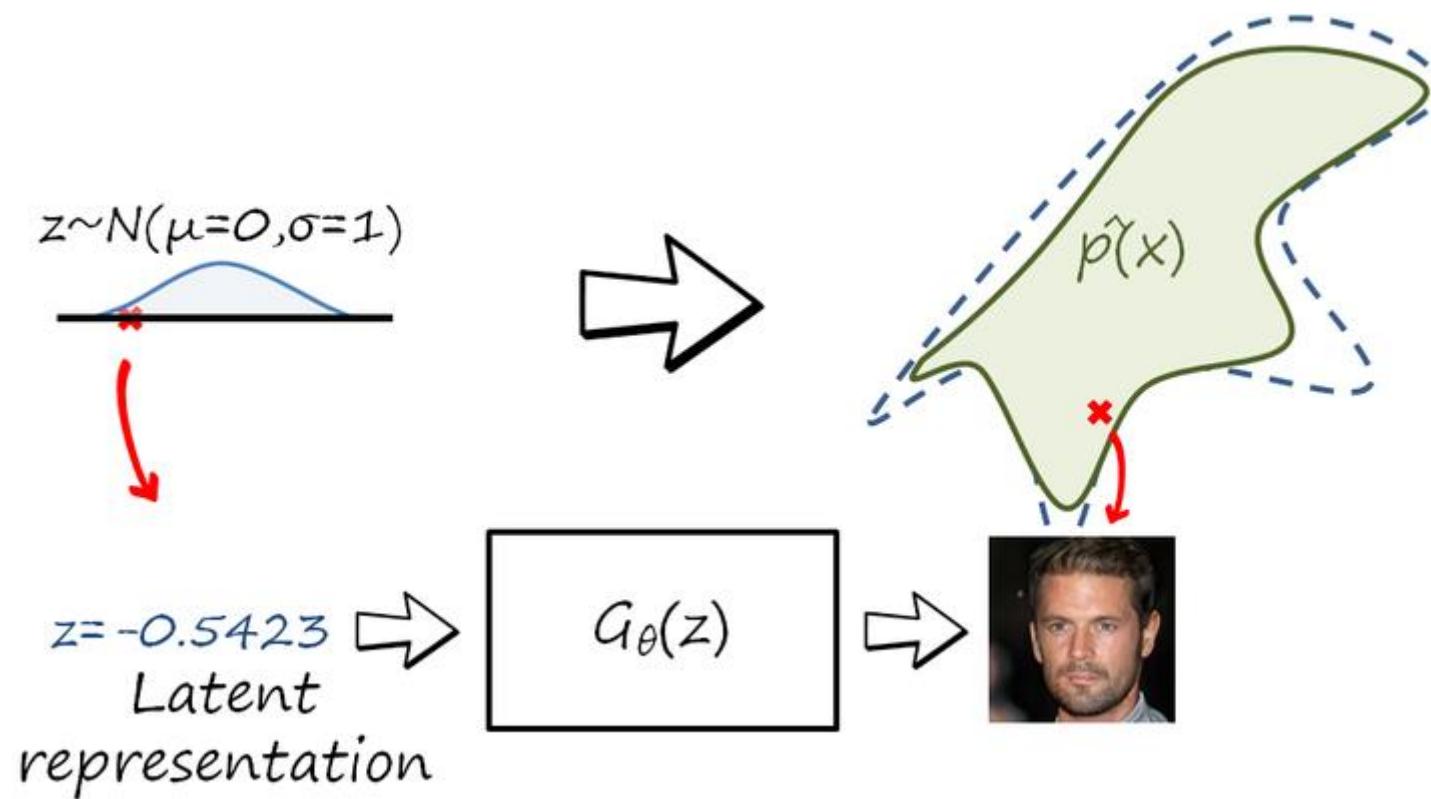


Generative modeling



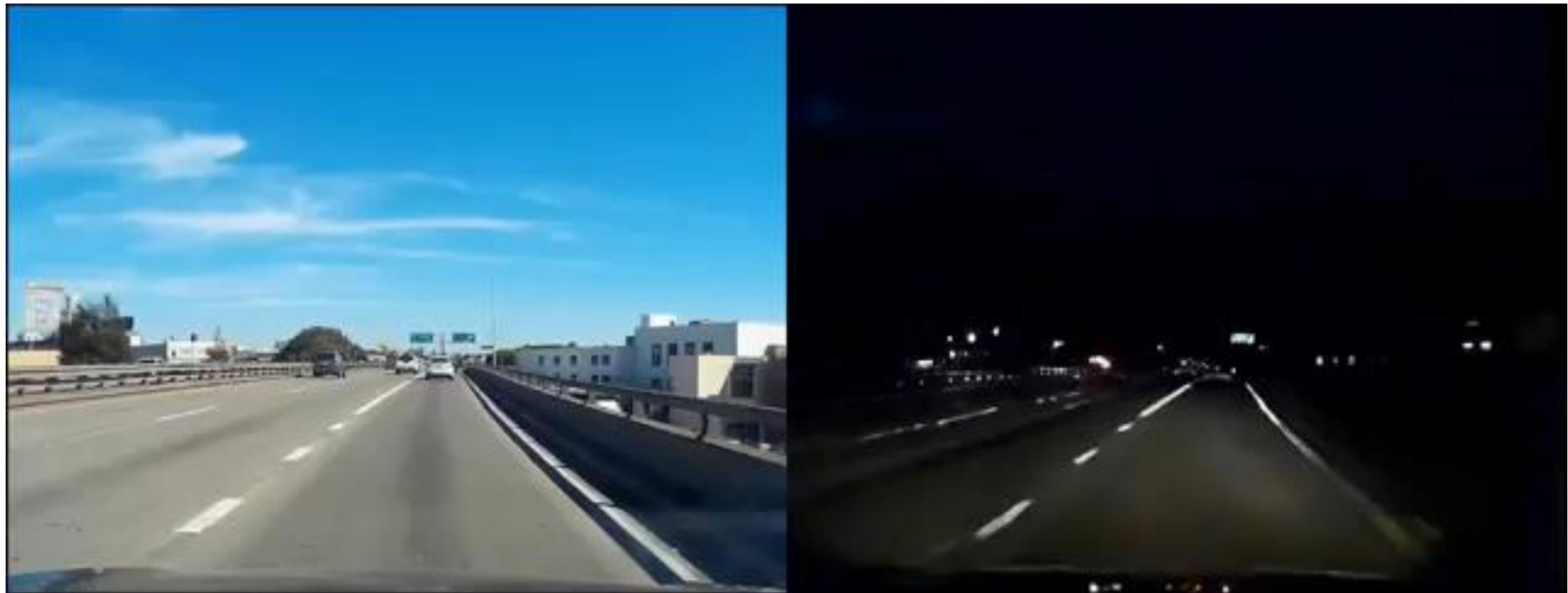
Courtesy of Luis Herranz

Generative Adversarial Networks



Courtesy of Luis Herranz

Unsupervised Image to Image Translation



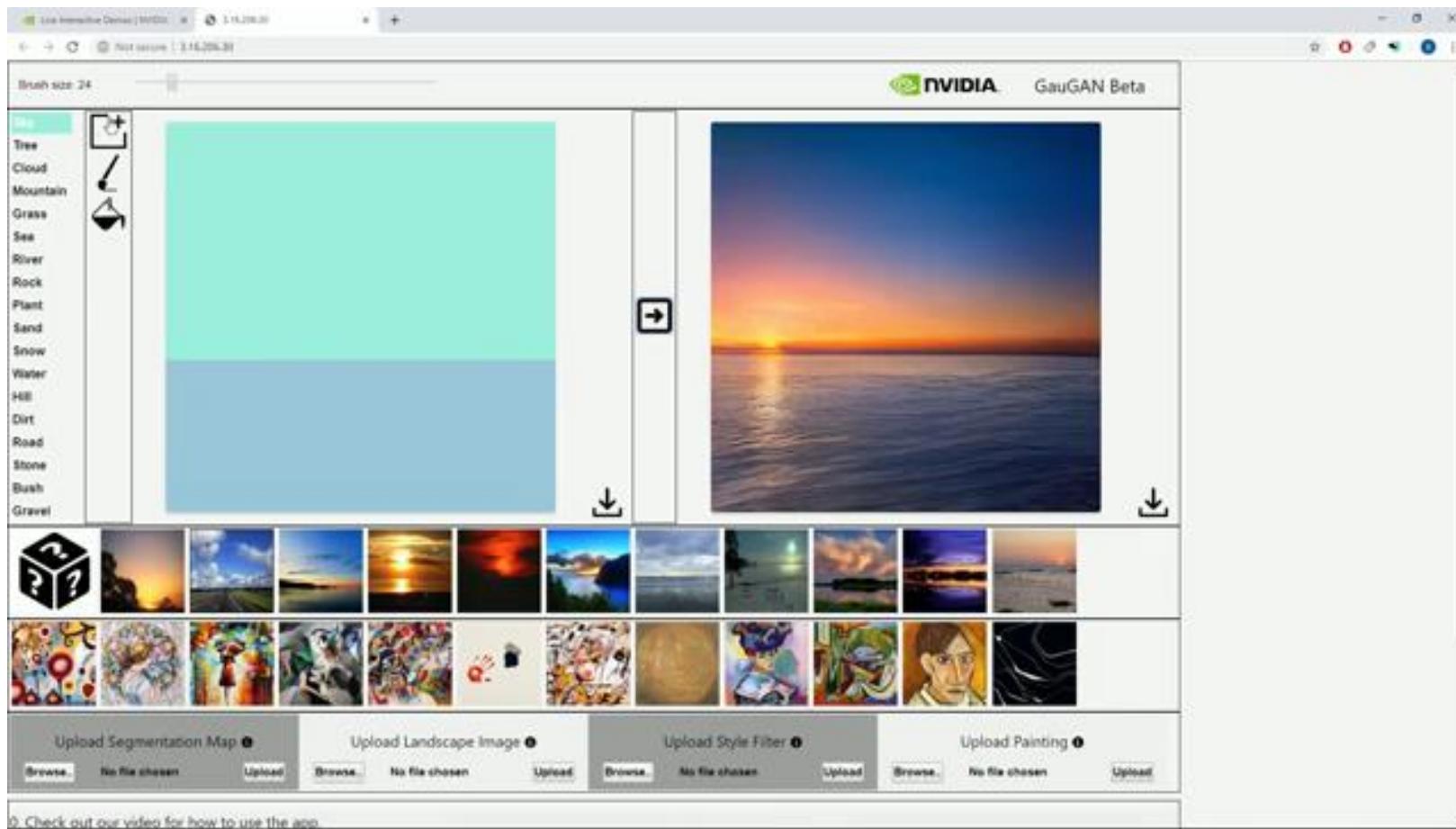
Liu et al (2017)

CycleGAN



Zhu et al (2017)

GauGan



NVidia

GAN

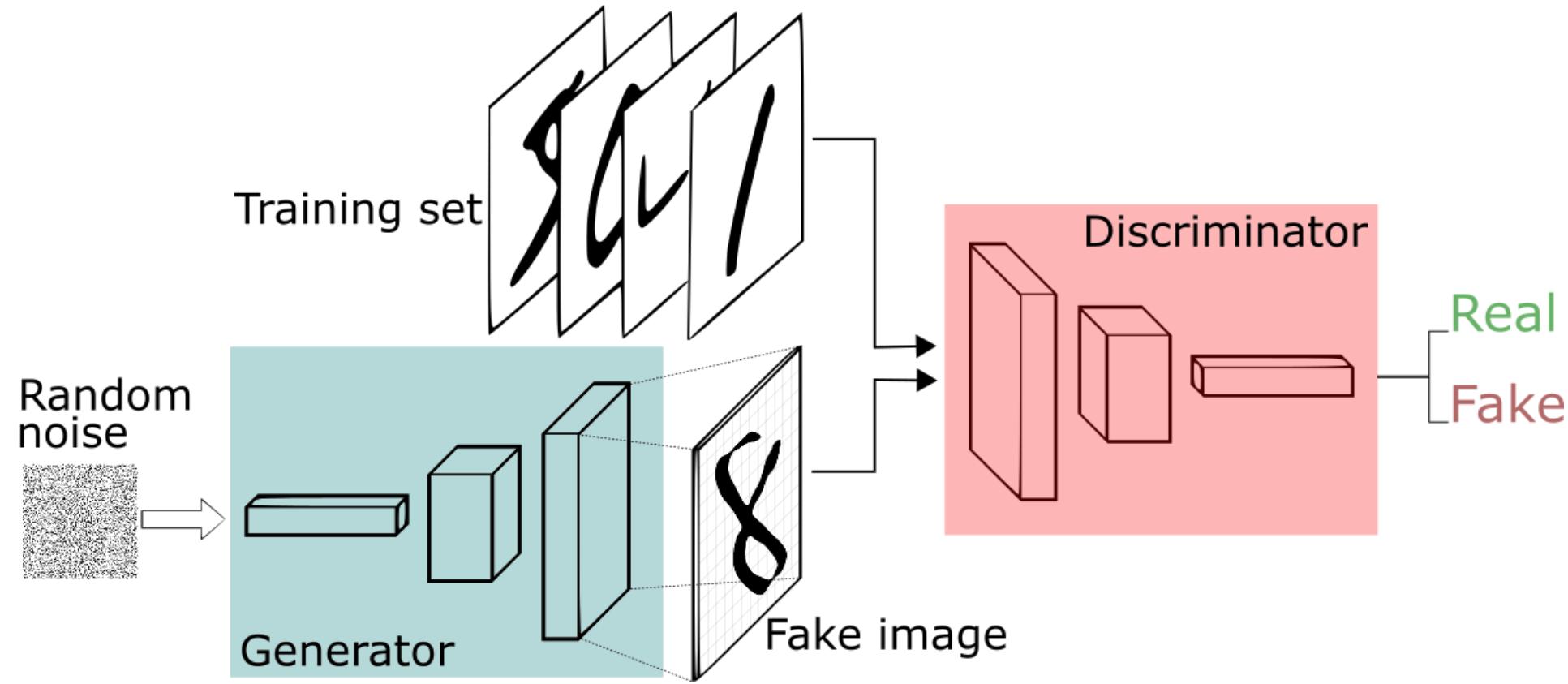
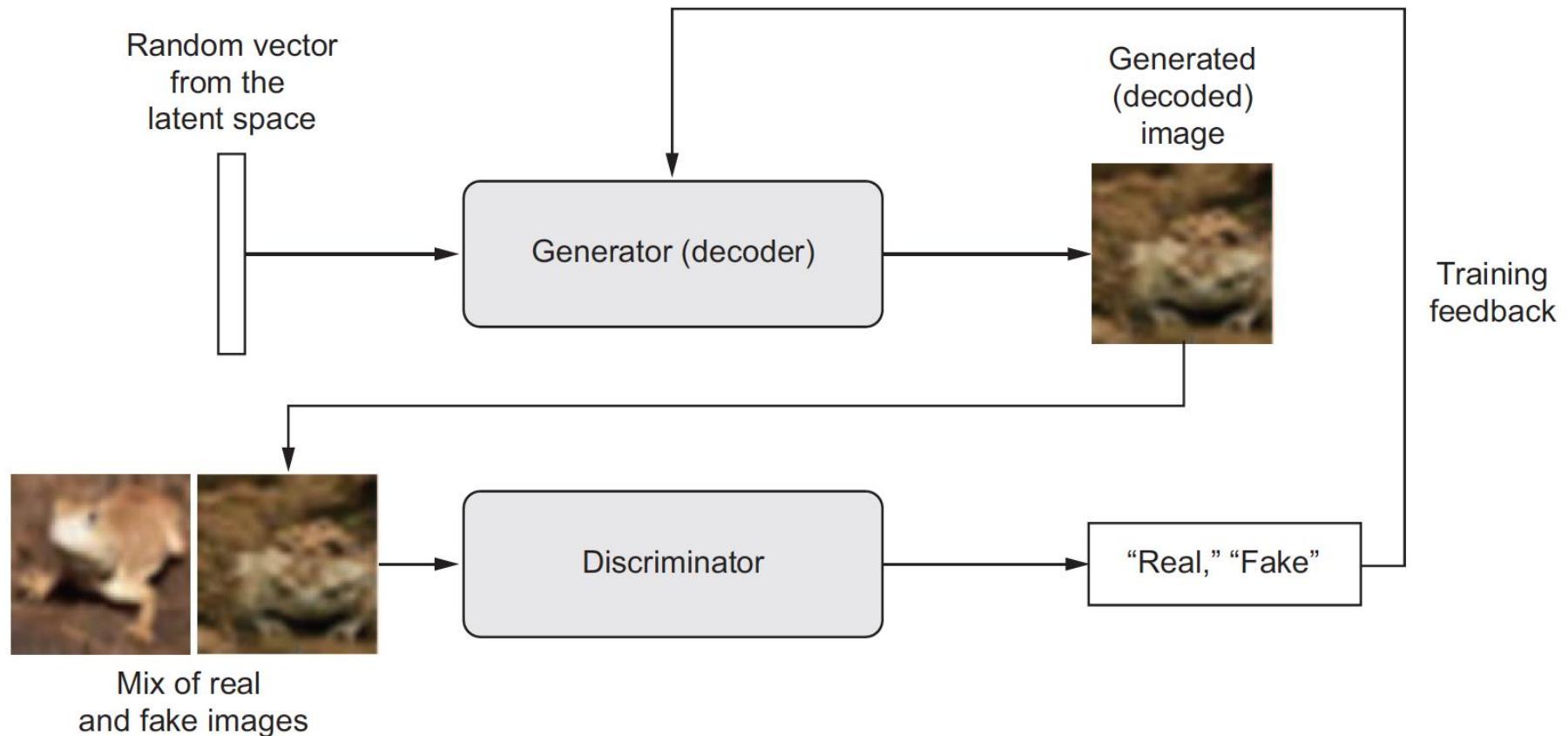


Image credit: Thalles Sil

GAN

- Generator network—Takes as input a random vector (a random point in the latent space), and decodes it into a synthetic image
- Discriminator network (or adversary)—Takes as input an image (real or synthetic), and predicts whether the image came from the training set or was created by the generator network.

GAN



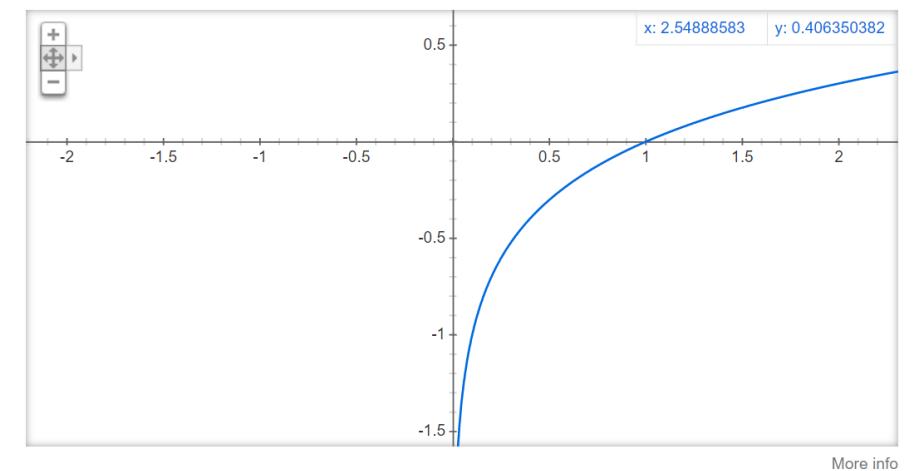
GAN: training procedure

Labels: $y_{real} = 1$
 $y_{gen} = 0$

Cost of discriminator

$$J^{(D)} = -\frac{1}{m_{real}} \sum_{i=1}^{m_{real}} y_{real}^{(i)} \cdot \log(D(x^{(i)})) - \frac{1}{m_{gen}} \sum_{i=1}^{m_{gen}} (1 - y_{gen}^{(i)}) \cdot \log(1 - D(G(z^{(i)})))$$

Graph for $\log(x)$



GAN: training procedure

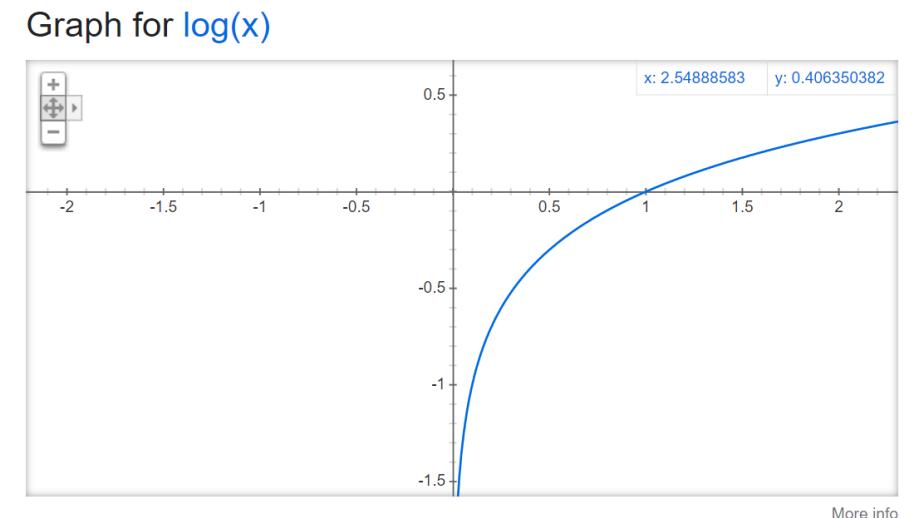
Labels: $y_{real} = 1$
 $y_{gen} = 0$

Cost of discriminator

$$J^{(D)} = -\frac{1}{m_{real}} \sum_{i=1}^{m_{real}} y_{real}^{(i)} \cdot \log(D(x^{(i)})) - \frac{1}{m_{gen}} \sum_{i=1}^{m_{gen}} (1 - y_{gen}^{(i)}) \cdot \log(1 - D(G(z^{(i)})))$$

Cost of generator

$$J^{(G)} = -J^{(D)} = \frac{1}{m_{gen}} \sum_{i=1}^{m_{gen}} \log(1 - D(G(z^{(i)})))$$



GAN: training procedure

Labels: $y_{real} = 1$
 $y_{gen} = 0$

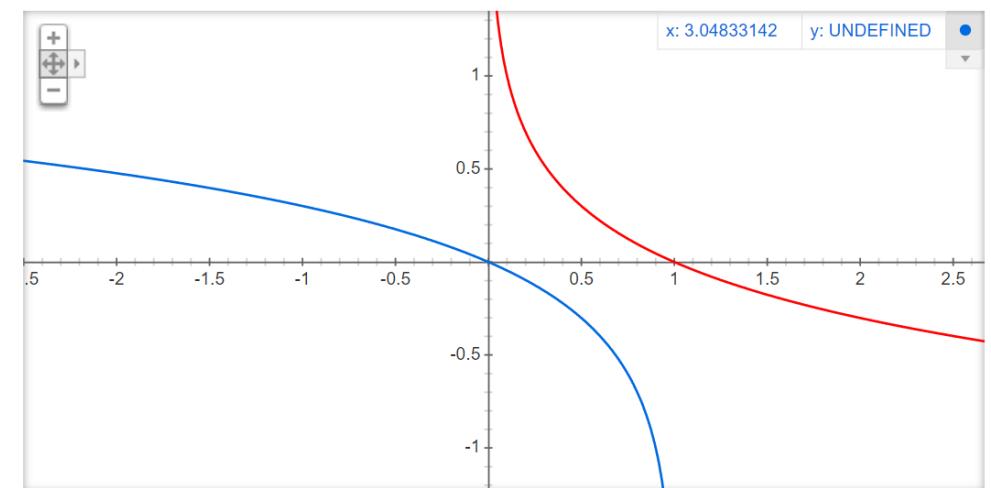
Cost of discriminator

$$J^{(D)} = -\frac{1}{m_{real}} \sum_{i=1}^{m_{real}} y_{real}^{(i)} \cdot \log(D(x^{(i)})) - \frac{1}{m_{gen}} \sum_{i=1}^{m_{gen}} (1 - y_{gen}^{(i)}) \cdot \log(1 - D(G(z^{(i)})))$$

Cost of generator

$$J^{(G)} = -\frac{1}{m_{gen}} \sum_{i=1}^{m_{gen}} \log(D(G(z^{(i)})))$$

Graph for $\log(1-x)$, $-\log(x)$



GAN: training procedure

Labels: $y_{\text{real}} = 1$
 $y_{\text{gen}} = 0$

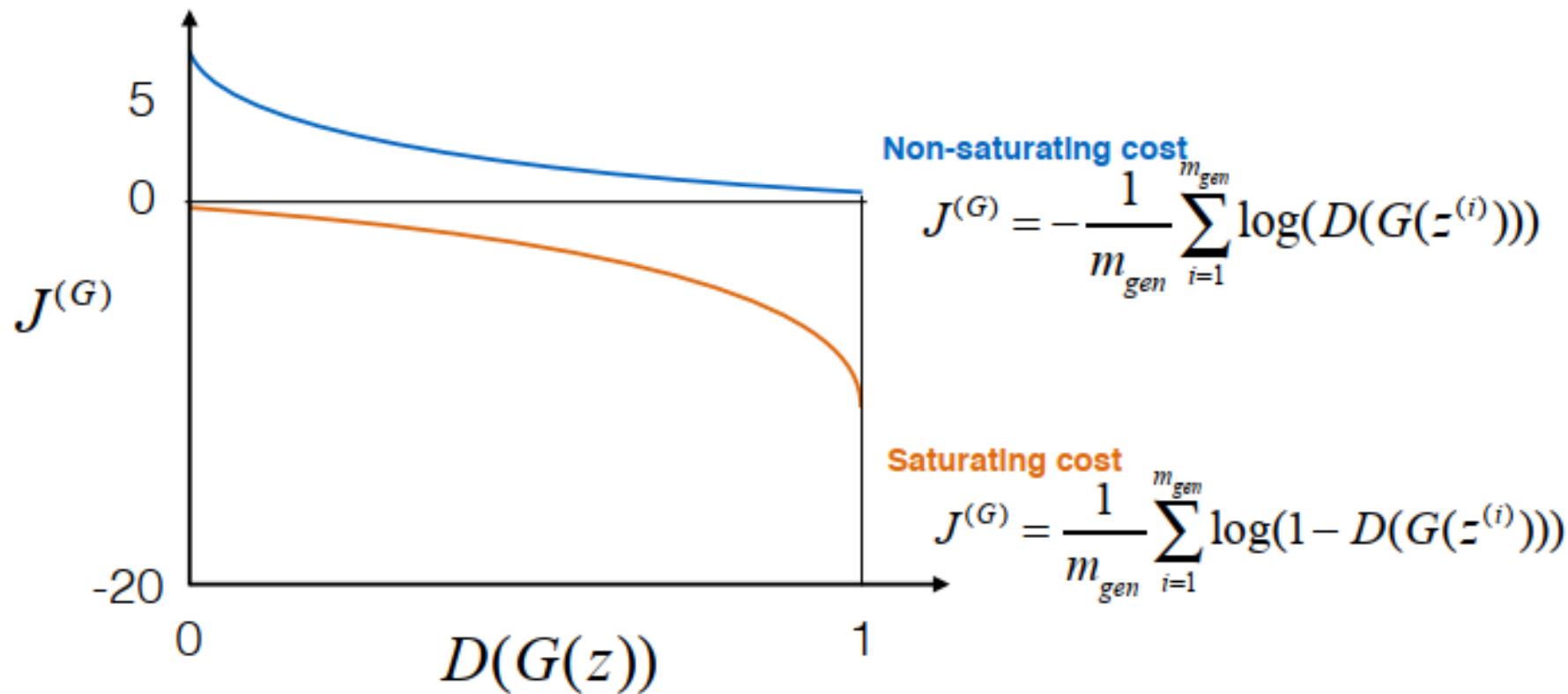


Table 1: Generator and discriminator loss functions. The main difference whether the discriminator outputs a probability (MM GAN, NS GAN, DRAGAN) or its output is unbounded (WGAN, WGAN GP, LS GAN, BEGAN), whether the gradient penalty is present (WGAN GP, DRAGAN) and where is it evaluated. We chose those models based on their popularity.

GAN	DISCRIMINATOR LOSS	GENERATOR LOSS
MM GAN	$\mathcal{L}_D^{GAN} = -\mathbb{E}_{x \sim p_d} [\log(D(x))] - \mathbb{E}_{\hat{x} \sim p_g} [\log(1 - D(\hat{x}))]$	$\mathcal{L}_G^{GAN} = \mathbb{E}_{\hat{x} \sim p_g} [\log(1 - D(\hat{x}))]$
NS GAN	$\mathcal{L}_D^{NSGAN} = -\mathbb{E}_{x \sim p_d} [\log(D(x))] - \mathbb{E}_{\hat{x} \sim p_g} [\log(1 - D(\hat{x}))]$	$\mathcal{L}_G^{NSGAN} = -\mathbb{E}_{\hat{x} \sim p_g} [\log(D(\hat{x}))]$
WGAN	$\mathcal{L}_D^{WGAN} = -\mathbb{E}_{x \sim p_d} [D(x)] + \mathbb{E}_{\hat{x} \sim p_g} [D(\hat{x})]$	$\mathcal{L}_G^{WGAN} = -\mathbb{E}_{\hat{x} \sim p_g} [D(\hat{x})]$
WGAN GP	$\mathcal{L}_D^{WGANGP} = \mathcal{L}_D^{WGAN} + \lambda \mathbb{E}_{\hat{x} \sim p_g} [(\nabla D(\alpha x + (1 - \alpha)\hat{x}) _2 - 1)^2]$	$\mathcal{L}_G^{WGANGP} = -\mathbb{E}_{\hat{x} \sim p_g} [D(\hat{x})]$
LS GAN	$\mathcal{L}_D^{LSGAN} = -\mathbb{E}_{x \sim p_d} [(D(x) - 1)^2] + \mathbb{E}_{\hat{x} \sim p_g} [D(\hat{x})^2]$	$\mathcal{L}_G^{LSGAN} = -\mathbb{E}_{\hat{x} \sim p_g} [(D(\hat{x}) - 1)^2]$
DRAGAN	$\mathcal{L}_D^{DRAGAN} = \mathcal{L}_D^{GAN} + \lambda \mathbb{E}_{\hat{x} \sim p_d + \mathcal{N}(0, c)} [(\nabla D(\hat{x}) _2 - 1)^2]$	$\mathcal{L}_G^{DRAGAN} = \mathbb{E}_{\hat{x} \sim p_g} [\log(1 - D(\hat{x}))]$
BEGAN	$\mathcal{L}_D^{BEGAN} = \mathbb{E}_{x \sim p_d} [x - AE(x) _1] - k_t \mathbb{E}_{\hat{x} \sim p_g} [\hat{x} - AE(\hat{x}) _1]$	$\mathcal{L}_G^{BEGAN} = \mathbb{E}_{\hat{x} \sim p_g} [\hat{x} - AE(\hat{x}) _1]$

GAN

- The optimization minimum isn't fixed, unlike what we had before.
- Normally, gradient descent consists of rolling down hills in a static loss landscape.
- But with a GAN, every step taken down the hill changes the entire landscape a little. It's a dynamic system where the optimization process is seeking not a minimum, but an equilibrium between two forces.

GAN progress



2014



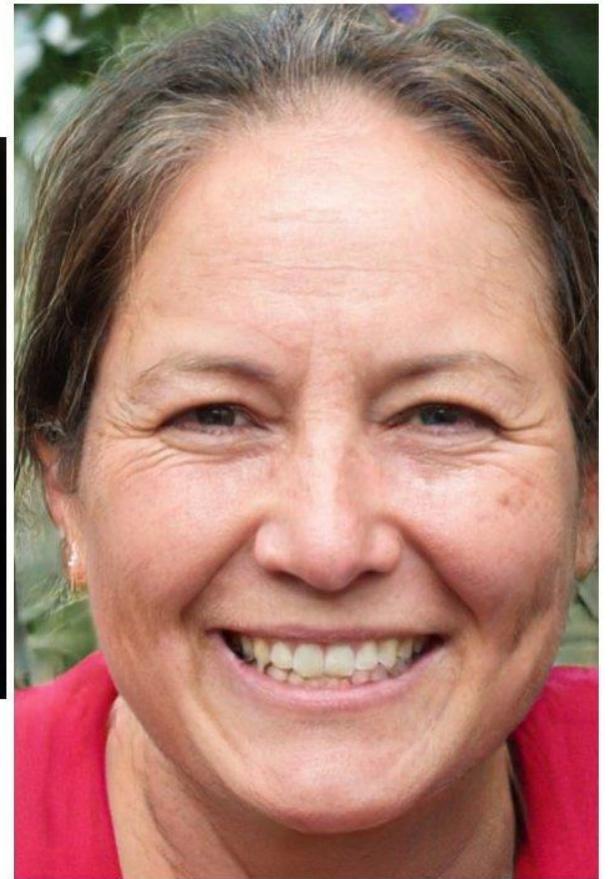
2015



2016



2017



2018

From Ian Goodfellow

Generative models progress



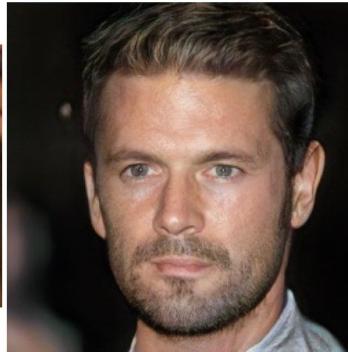
2014



2015



2016



2017



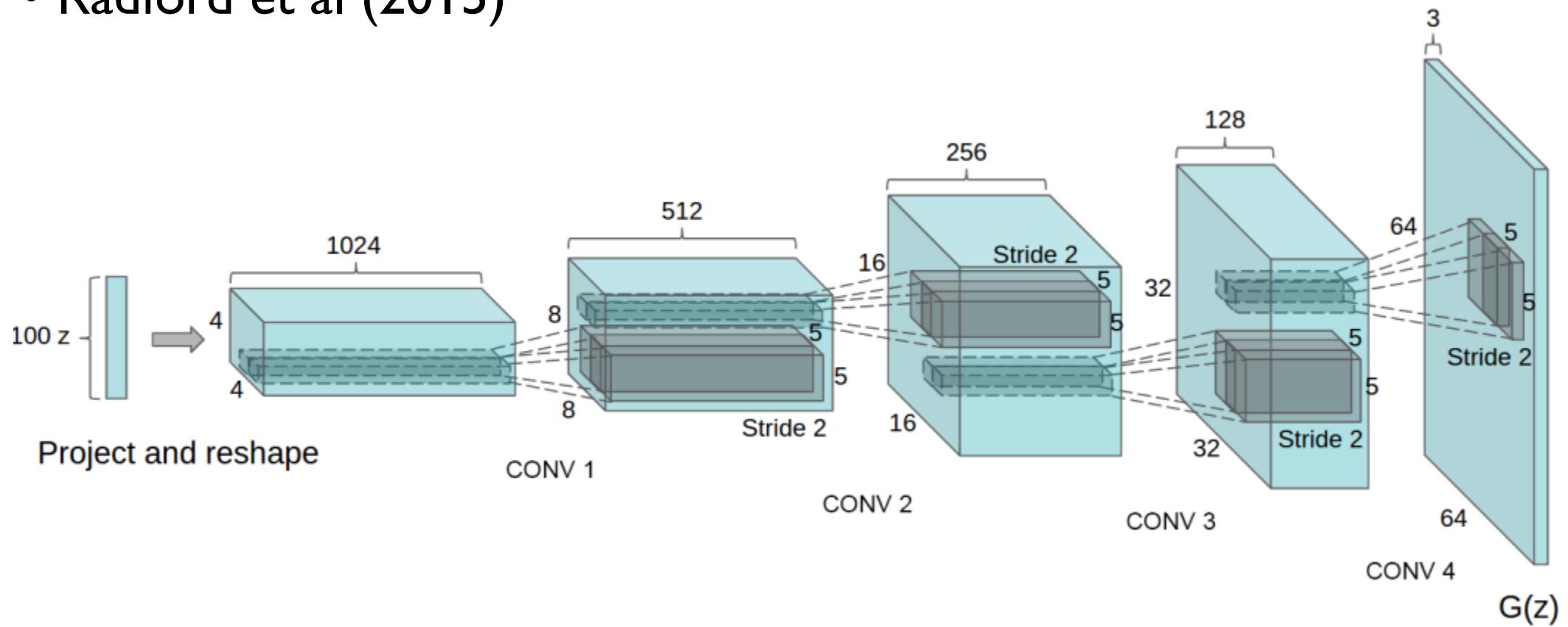
2018



2022: Stable Diffusion

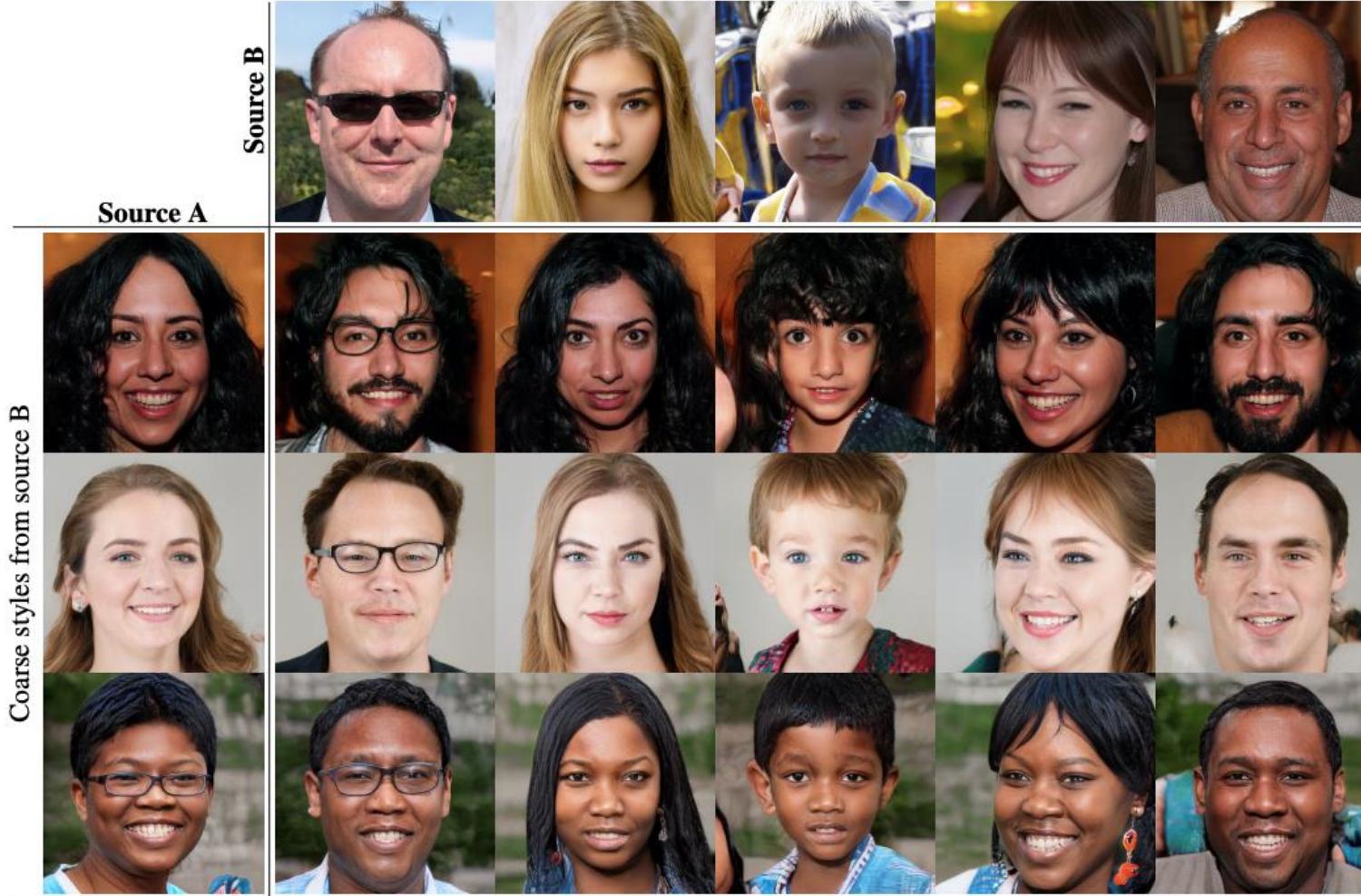
DCGAN

- Radford et al (2015)



StyleGAN

[Read more](#)



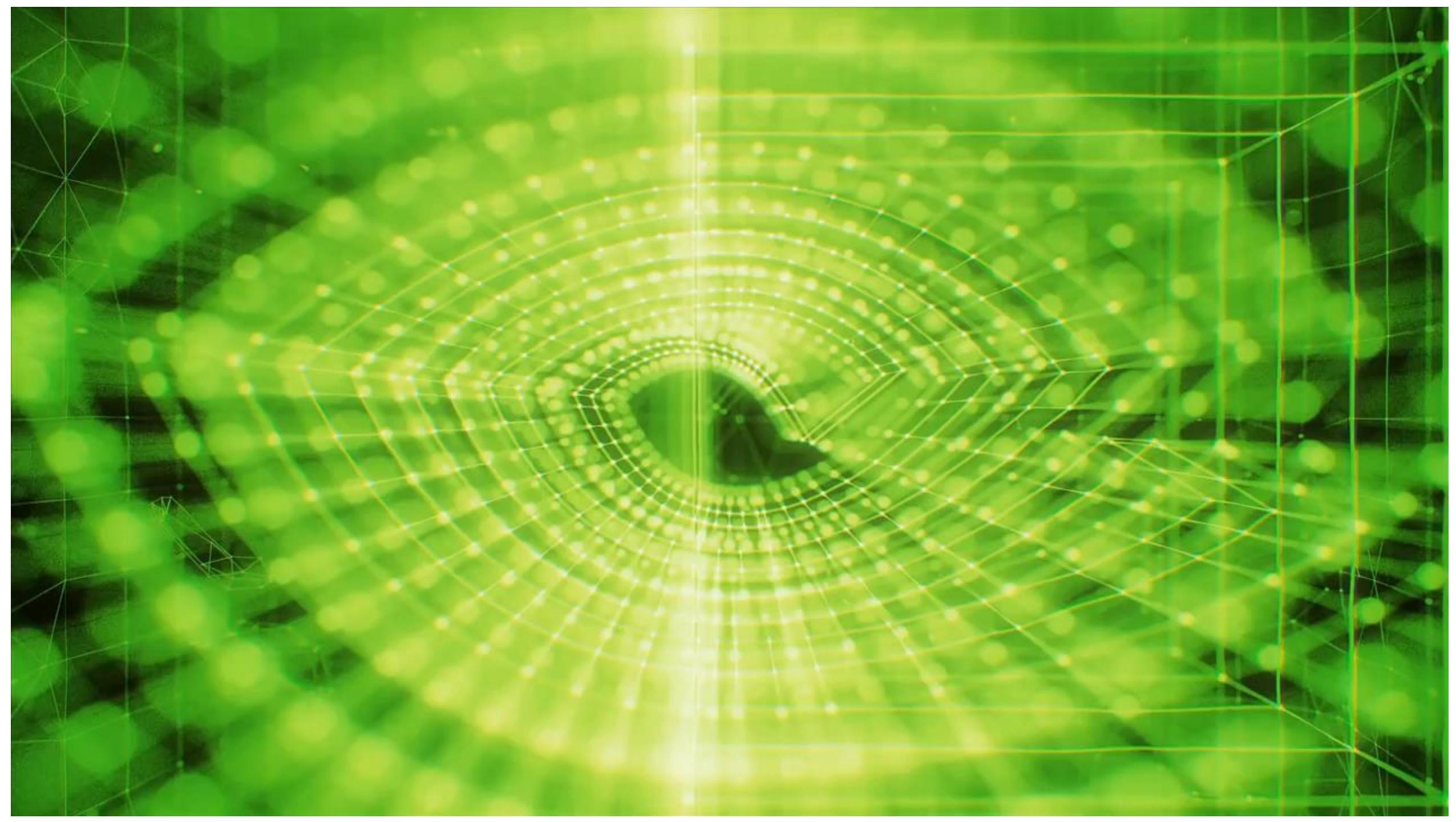
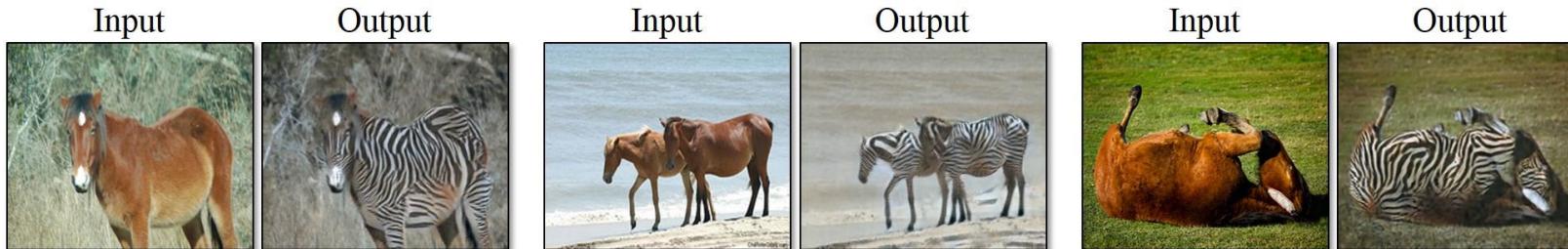


Image to Image Translation



horse → zebra



zebra → horse



apple → orange



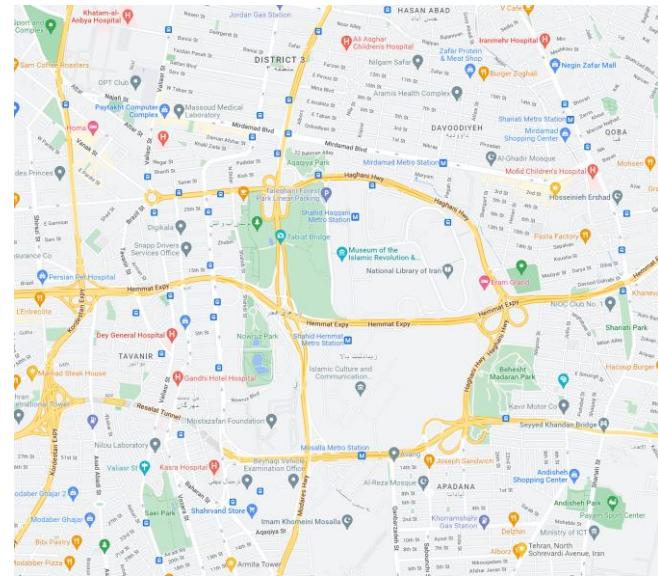
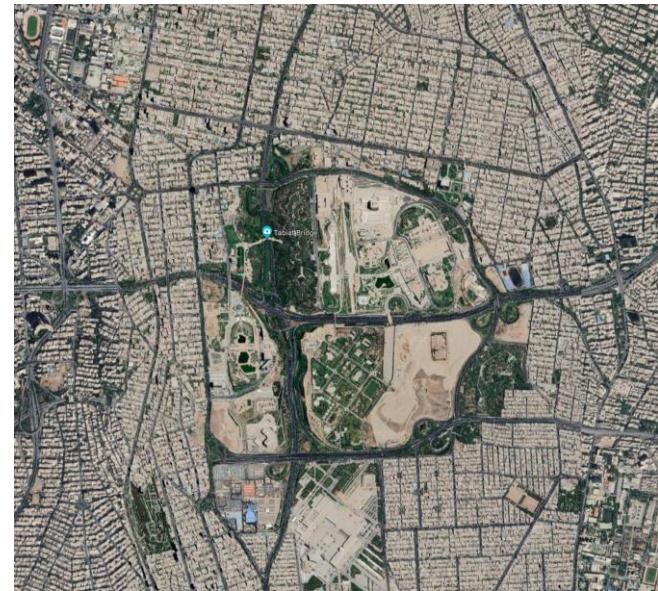
orange → apple

Image to Image Translation



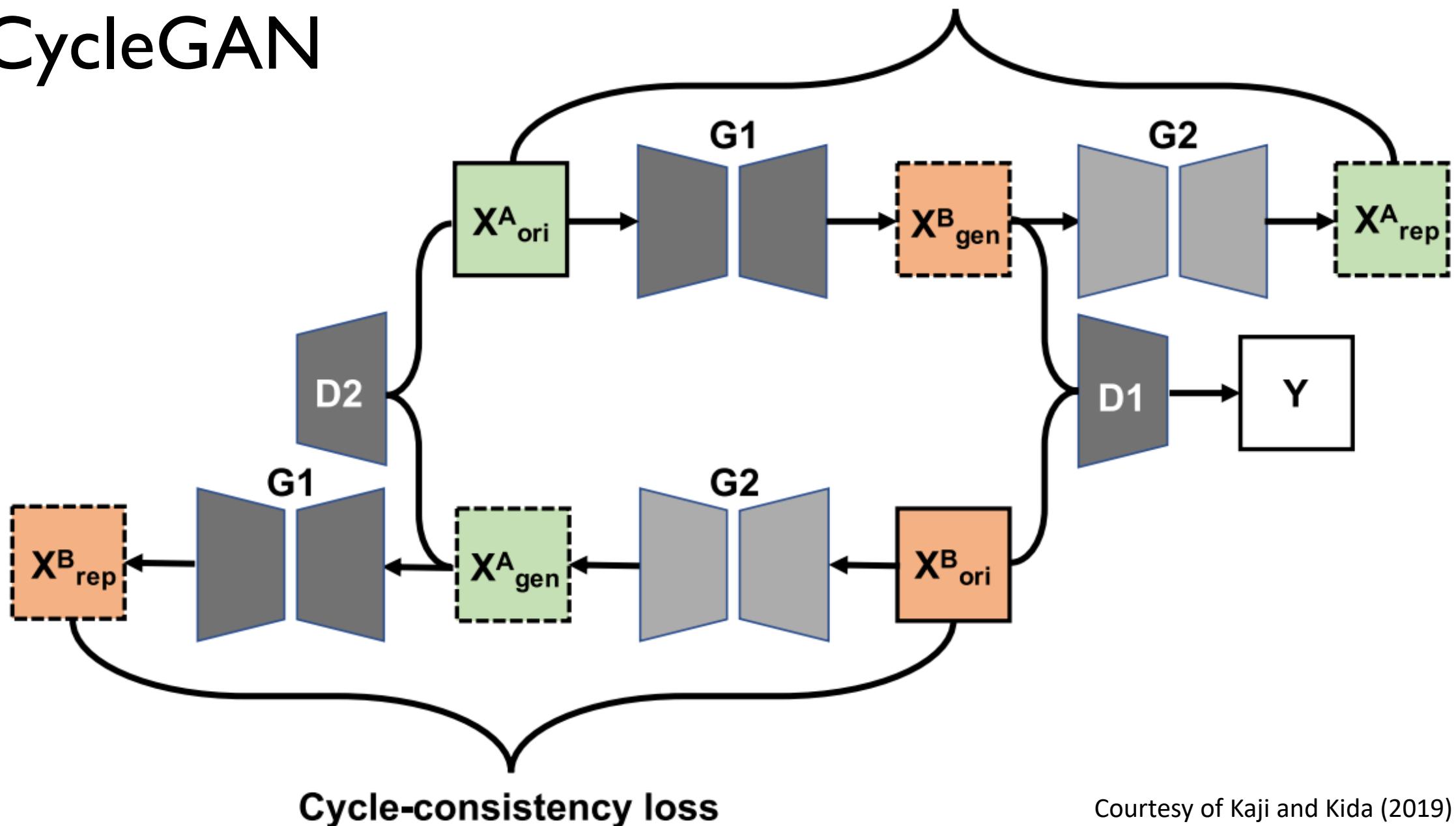
Figure 8: Example results on Google Maps at 512x512 resolution (model was trained on images at 256 × 256 resolution, and run convolutionally on the larger images at test time). Contrast adjusted for clarity.

Pix2pix



CycleGAN

Cycle-consistency loss

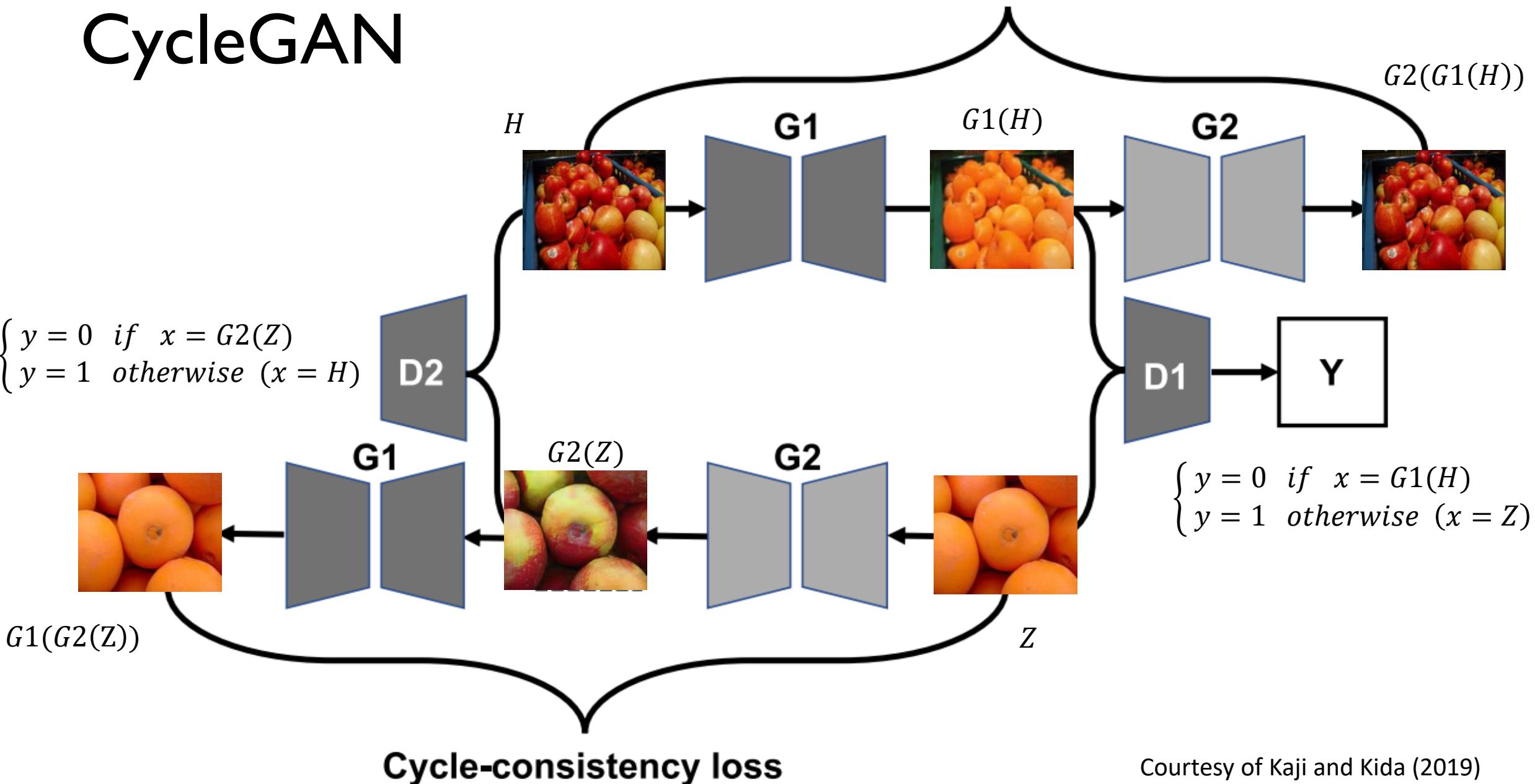


Cycle-consistency loss

Courtesy of Kaji and Kida (2019)

CycleGAN

Cycle-consistency loss



Courtesy of Kaji and Kida (2019)

CycleGAN

$$J^{(D1)} = -\frac{1}{m_{real}} \sum_{i=1}^{m_{real}} \log(D1(z^{(i)})) - \frac{1}{m_{gen}} \sum_{i=1}^{m_{gen}} \log(1 - D1(G1(H^{(i)})))$$

$$J^{(G1)} = -\frac{1}{m_{gen}} \sum_{i=1}^{m_{gen}} \log(D1(G1(H^{(i)})))$$

$$J^{(D2)} = -\frac{1}{m_{real}} \sum_{i=1}^{m_{real}} \log(D2(h^{(i)})) - \frac{1}{m_{gen}} \sum_{i=1}^{m_{gen}} \log(1 - D2(G2(Z^{(i)})))$$

$$J^{(G2)} = -\frac{1}{m_{gen}} \sum_{i=1}^{m_{gen}} \log(D2(G2(Z^{(i)})))$$

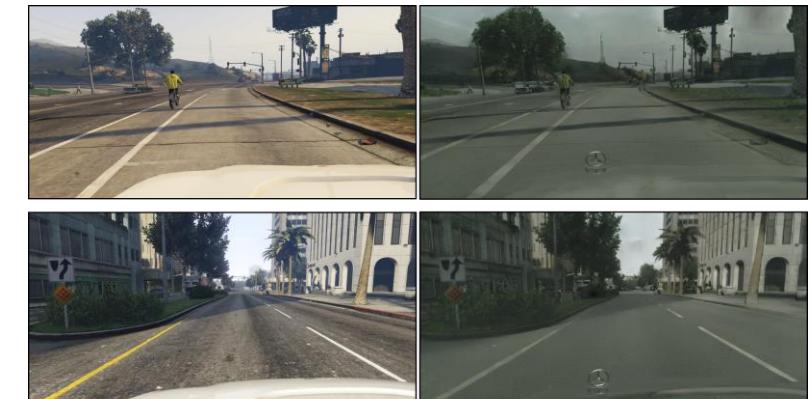
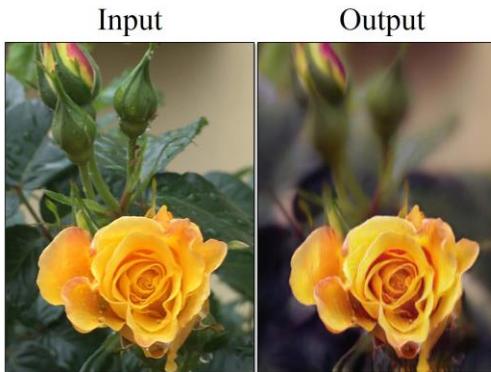
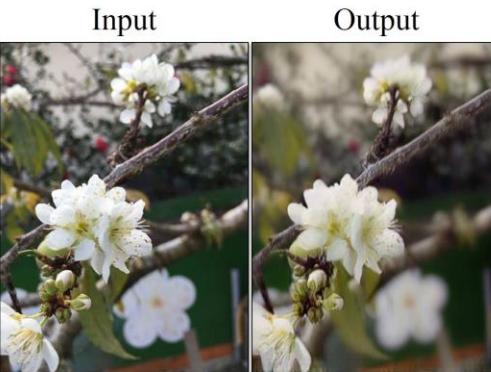
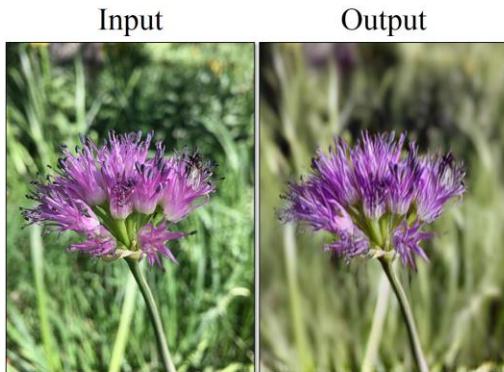
$$J = J^{(D1)} + J^{(G1)} + J^{(D2)} + J^{(G2)} + \lambda J^{cycle}$$

$$J^{cycle} = \frac{1}{m_{gen}} \sum_{i=1}^{m_{gen}} \| G2(G1(H^{(i)}) - H^{(i)} \|_1 + \frac{1}{m_{gen}} \sum_{i=1}^{m_{gen}} \| G1(G2(Z^{(i)}) - Z^{(i)} \|_1$$

Read more

More examples for CycleGAN

- <https://junyanz.github.io/CycleGAN/>



GTA → Cityscapes

Mode collapse

It consists in the **generator “collapsing”** and always generating a single image for every possible latent vector fed as input.

Read more about GANs, their variants (super-resolution, face inpainting, etc), and their common problems.

<https://developers.google.com/machine-learning/gan>

Generative models

Prisma



Generative models

DeepDream



ChatGPT

MP

What is the best way to evaluate students in the Machine Learning course?



There are several ways to evaluate students in a machine learning course, and the best approach will depend on the specific goals and objectives of the course, as well as the level of the students. Some common methods of evaluation include:

1. Examinations: Written exams can test students' understanding of key concepts and techniques in machine learning. These can be in the form of multiple-choice questions, short answer questions, or essay questions.
2. Homework assignments: Homework assignments can help students apply what they have learned in class to specific problems and can be a good way to assess their progress.
3. Projects: Students can complete a machine learning project as a way to demonstrate their understanding of the material and their ability to apply it in a practical setting. Projects can be individual or group-based, and can involve building a machine learning model to solve a specific problem.
4. Presentations: Students can give presentations on a machine learning topic of their choice as a way to demonstrate their understanding and ability to communicate technical concepts to a non-technical audience.
5. Quizzes: Quizzes can be used to assess students' understanding of key concepts in machine learning and can be given in class or as part of an online course.

Ultimately, the best approach to evaluating students in a machine learning course will depend on the specific goals and objectives of the course, as well as the level and experience of the students. It is often helpful to use a combination of different evaluation methods to get a comprehensive understanding of students' progress and understanding of the material.



ChatGPT

MP

How does snow affect the traffic?



Snow can have a significant impact on traffic, as it can make roads slippery and hazardous to drive on. Snowfall can also reduce visibility, making it more difficult for drivers to see other vehicles or obstacles in the road. In addition, snow can accumulate on the roads and cause them to become narrow or blocked, which can lead to delays or accidents. Snow can also cause problems for vehicles, such as reduced traction or reduced braking ability.

Like Dislike

To reduce the impact of snow on traffic, it is important for drivers to take precautions when driving in snowy conditions. This may include driving at a slower speed, increasing the distance between your vehicle and other vehicles, and using appropriate tires for the conditions. It is also important to be prepared for delays and to allow extra time to reach your destination. In addition, it is a good idea to have an emergency kit in your vehicle in case you become stranded in the snow.

ChatGPT

MP

is it possible to grow watermelon on Jupiter?



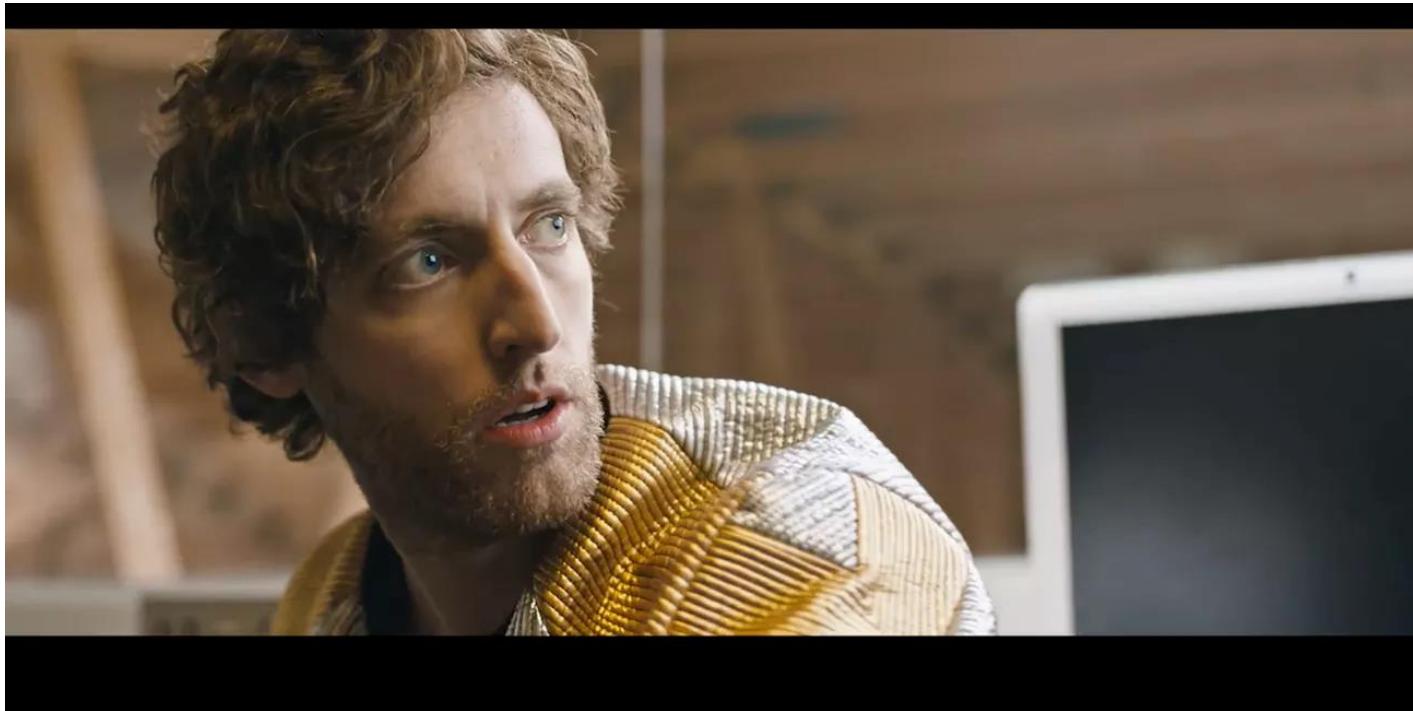
It is not currently possible to grow watermelon or any other plants on Jupiter, as the planet does not have the necessary conditions for life as we know it. Jupiter is a gas giant with no solid surface, and it has a thick atmosphere made up of hydrogen and helium. The temperature and pressure in the atmosphere of Jupiter are also much too high to support life, making it inhospitable to plants or animals.



In addition, Jupiter is located much too far from the sun to receive the necessary amount of sunlight for plants to grow. Watermelons and other plants need sunlight to photosynthesize and produce the energy they need to grow and survive. Without sufficient sunlight, it would be impossible for watermelons or any other plants to grow on Jupiter.

Generative models

Sunspring



Generative models

This AI-generated portrait just sold at auction for \$432K



Stable Diffusion

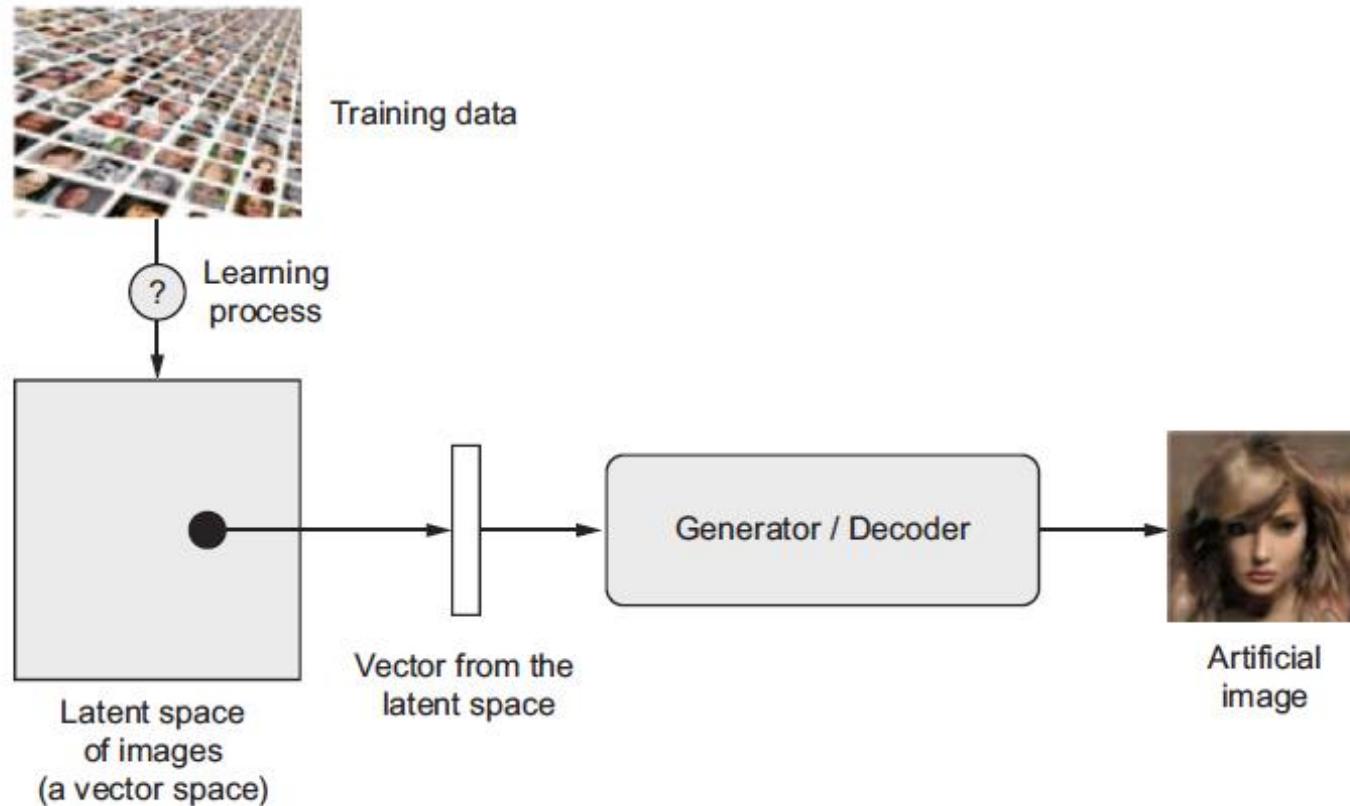


monkeys driving bikes on highway

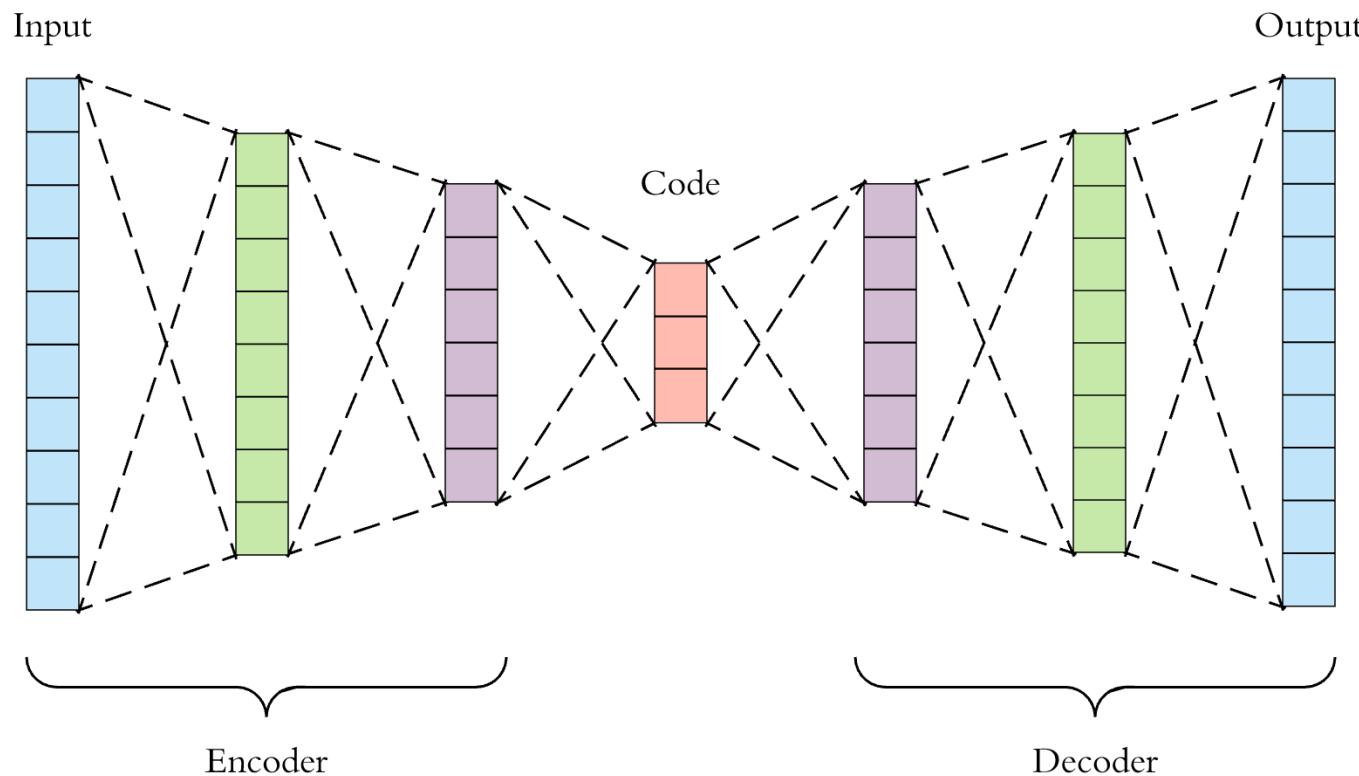


golden macbook

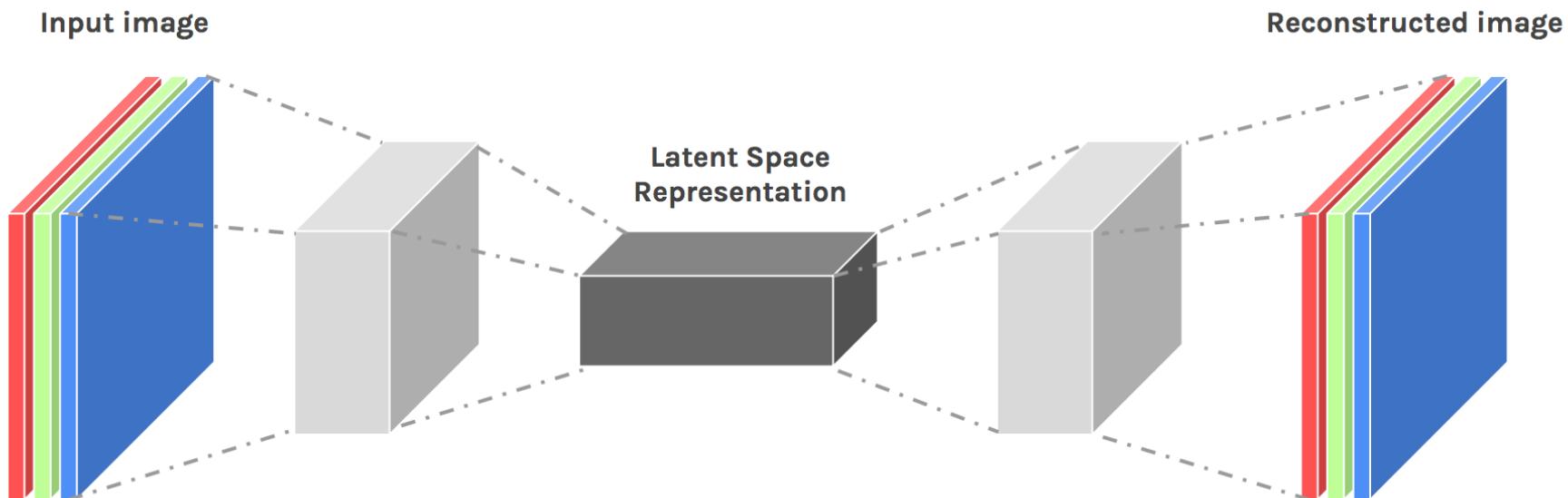
Variational autoencoders



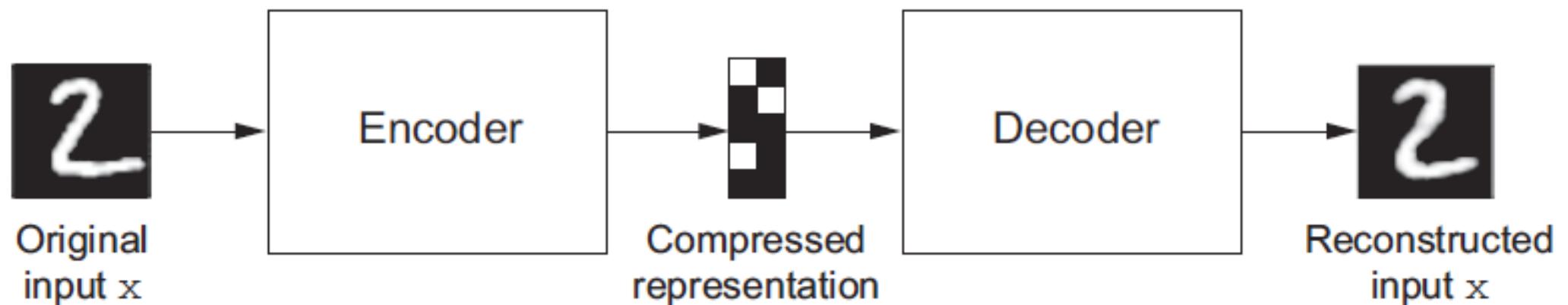
Autoencoders



Autoencoders



Autoencoders



Autoencoders

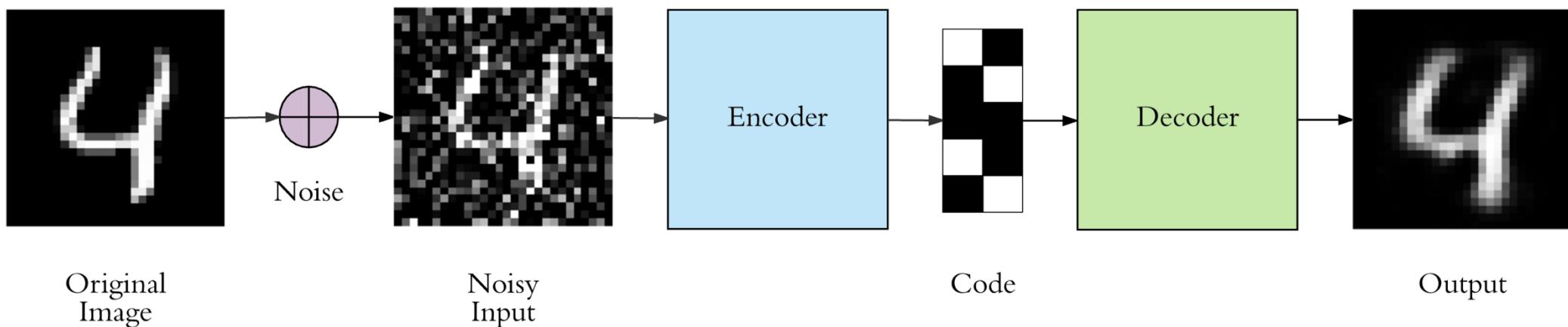
Training an autoencoder on the
MNIST dataset, and visualizing
the encodings from a 2D latent space



Figure by Irhum Shafkat

Variational autoencoders

Denoising autoencoder



Variational autoencoders

Advantage over autoencoders (that makes them suitable for generative models)

- Latent space is continuous

This is achieved by making the encoder not output an encoding vector of size n , rather, outputting two vectors of size n :

- a vector of means μ
- a vector of standard deviations σ

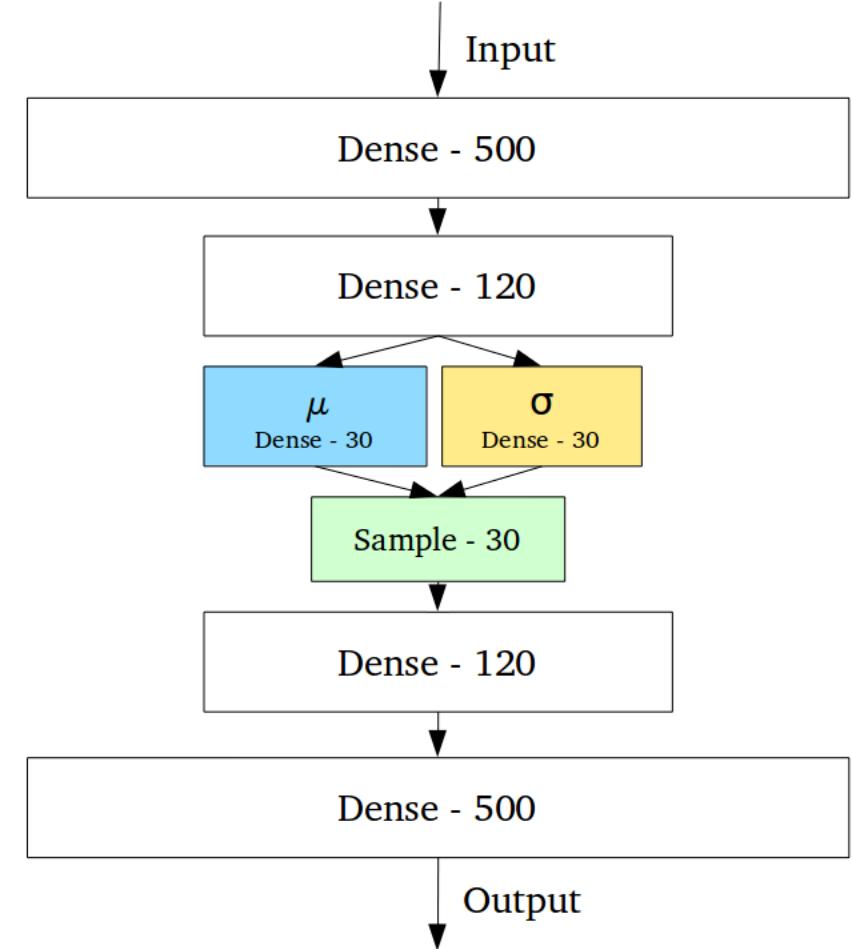


Figure by Irhum Shafkat

Concept space

Semantic space of embeddings is a specific case

Given a latent space of representations, certain directions may encode interesting axes of variation in the original data.

- If \mathbf{z} is the representation for a face, there might be a smile vector \mathbf{s} , then $\mathbf{z} + \mathbf{s}$ is the same face with smile
- Same for turning a male face to female
- Or adding sunglasses to a face

Concept space

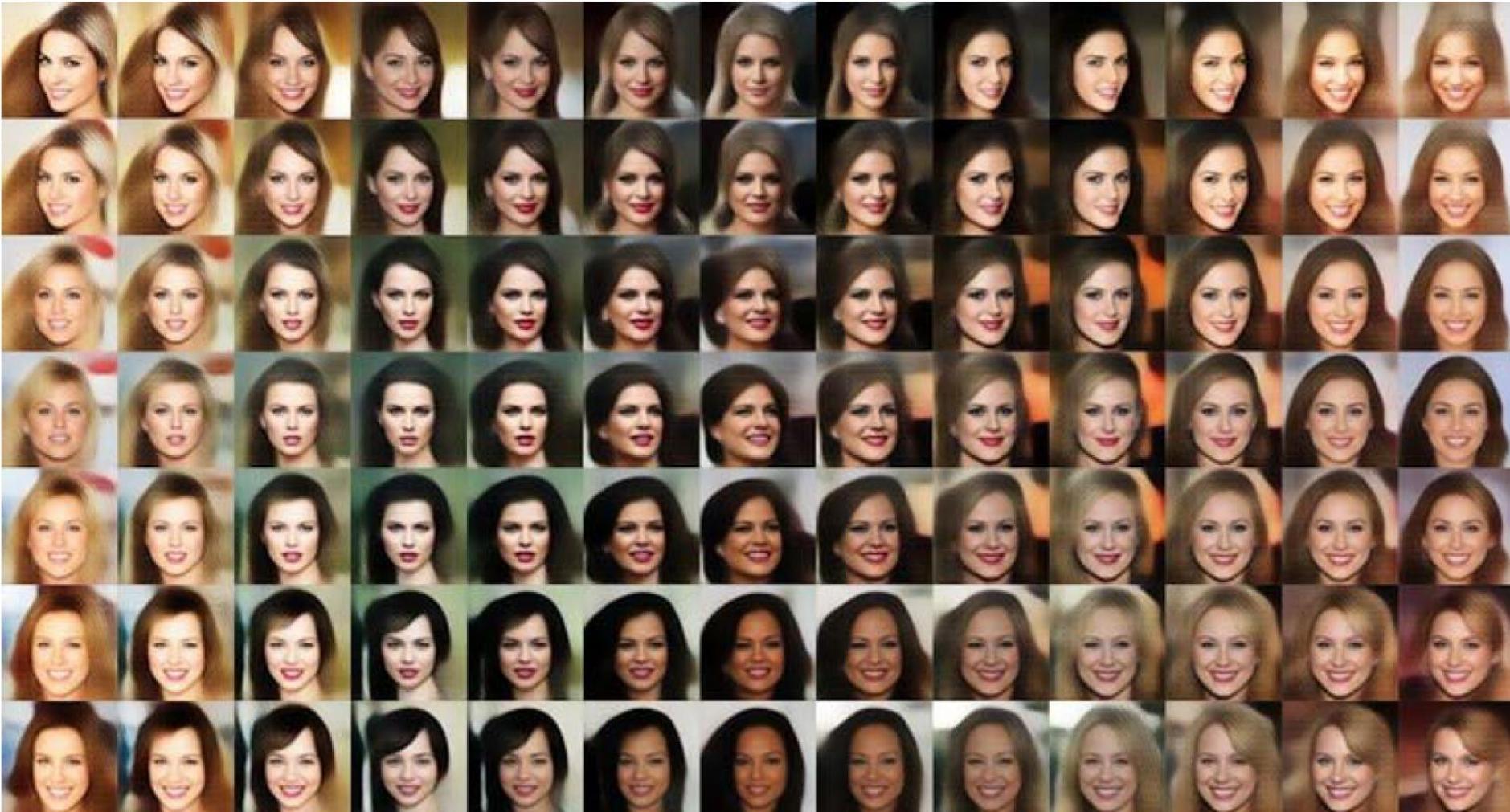


Figure 8.10 A continuous space of faces generated by Tom White using VAEs

Concept space

Smile vector

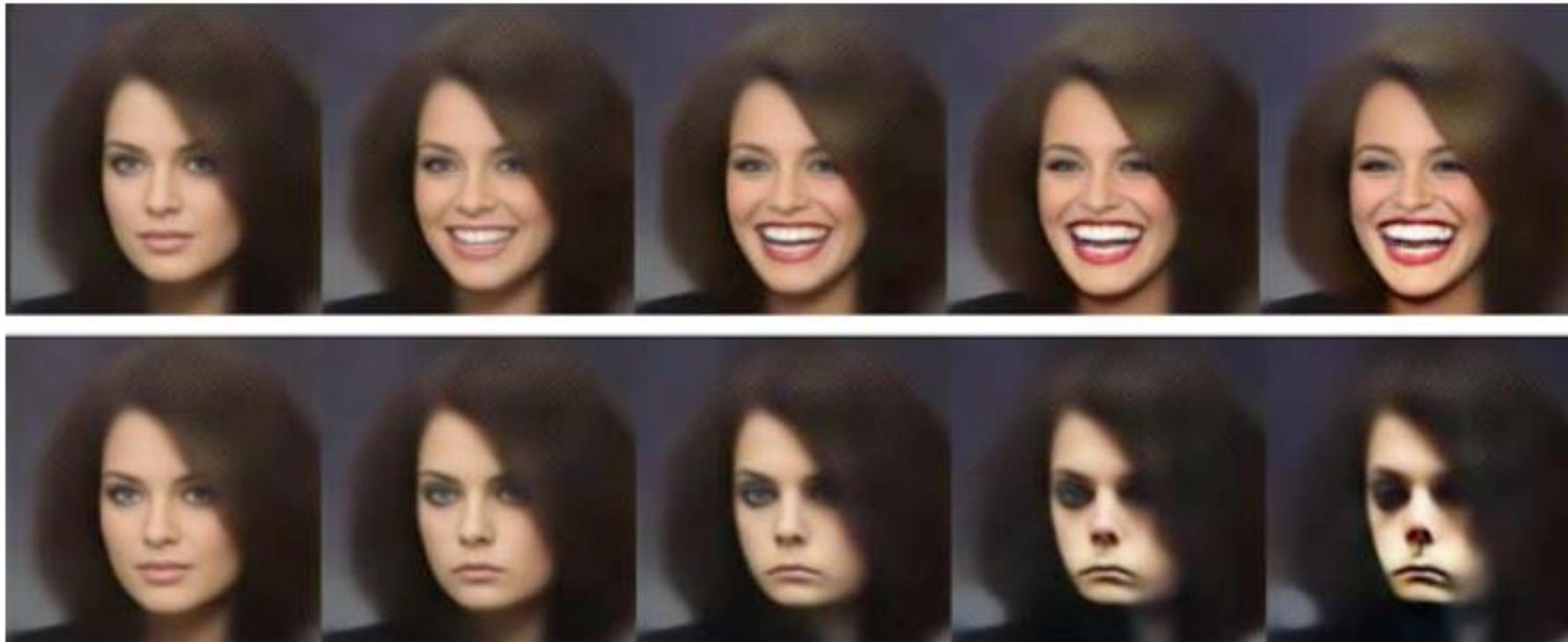
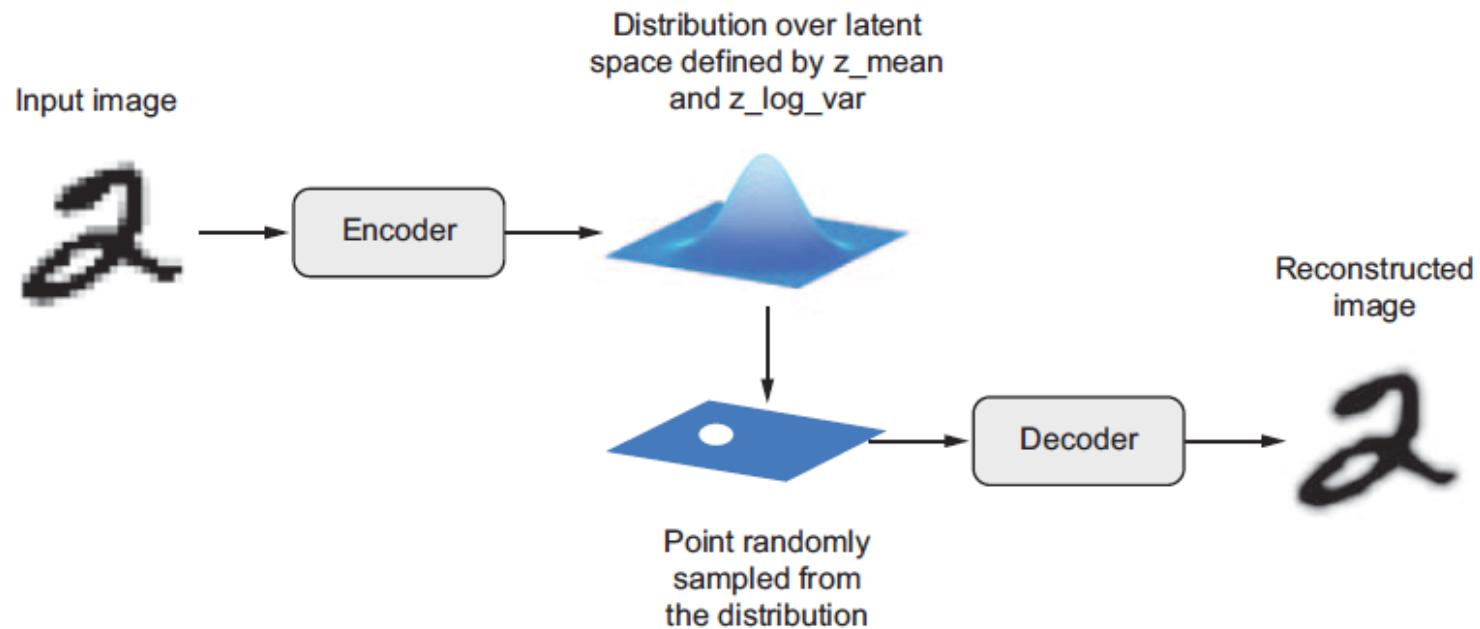
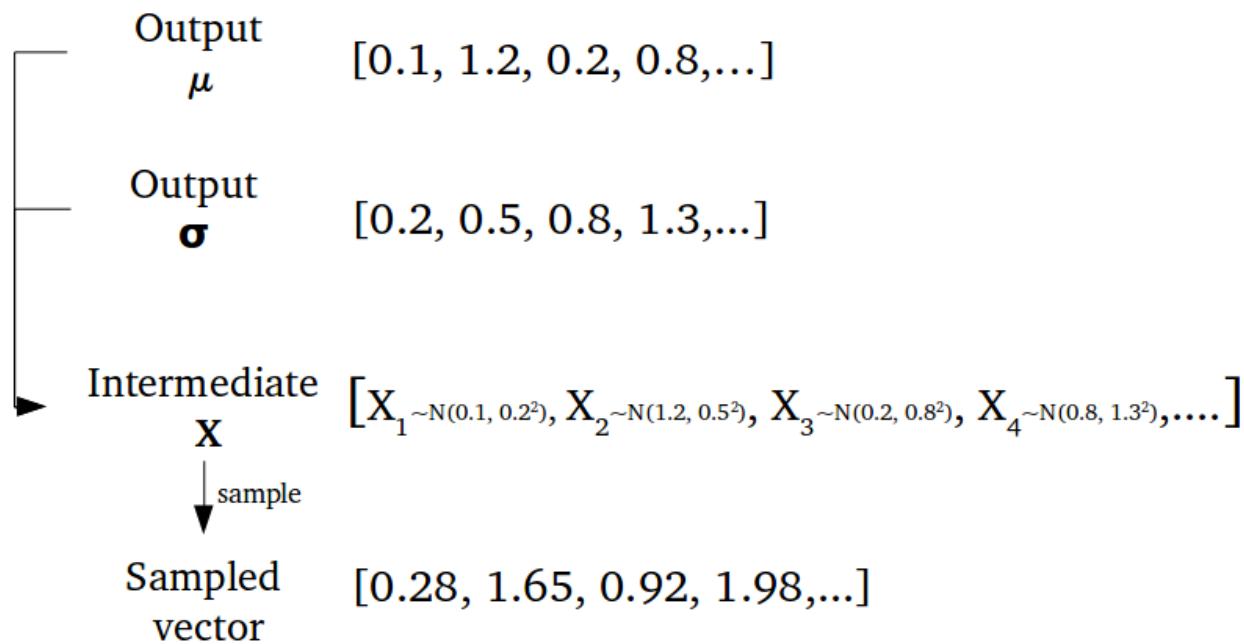


Image from Tom White (Victoria University School of Design in New Zealand)

Variational autoencoders



Variational autoencoders



Variational autoencoders

- An encoder module turns the input samples `input_img` into two parameters in a latent space of representations, `z_mean` and `z_log_variance`.
- You randomly sample a point `z` from the latent normal distribution that's assumed to generate the input image, via
$$z = z_mean + \exp(z_log_variance) * \epsilon$$
where `epsilon` is a random tensor of small values.
- A decoder module maps this point in the latent space back to the original input image.

Variational autoencoders

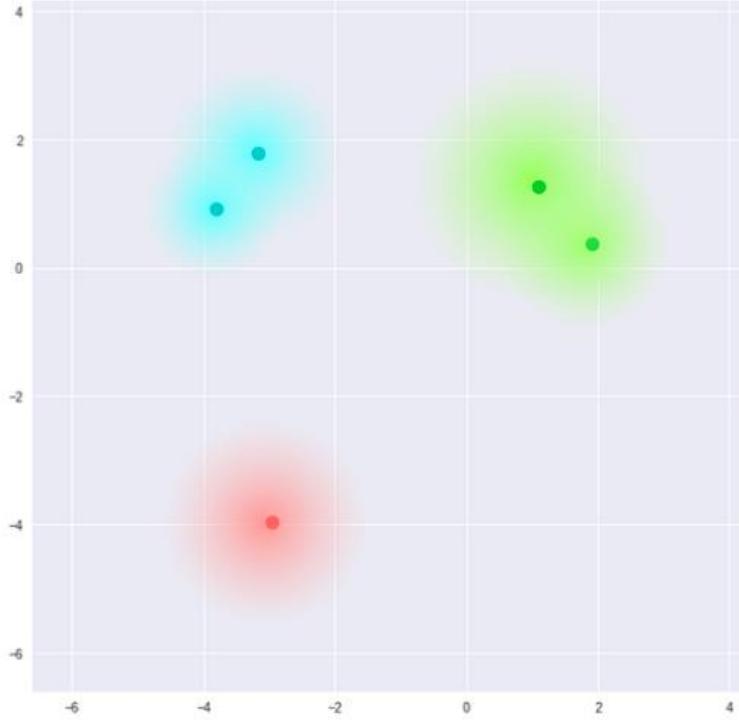
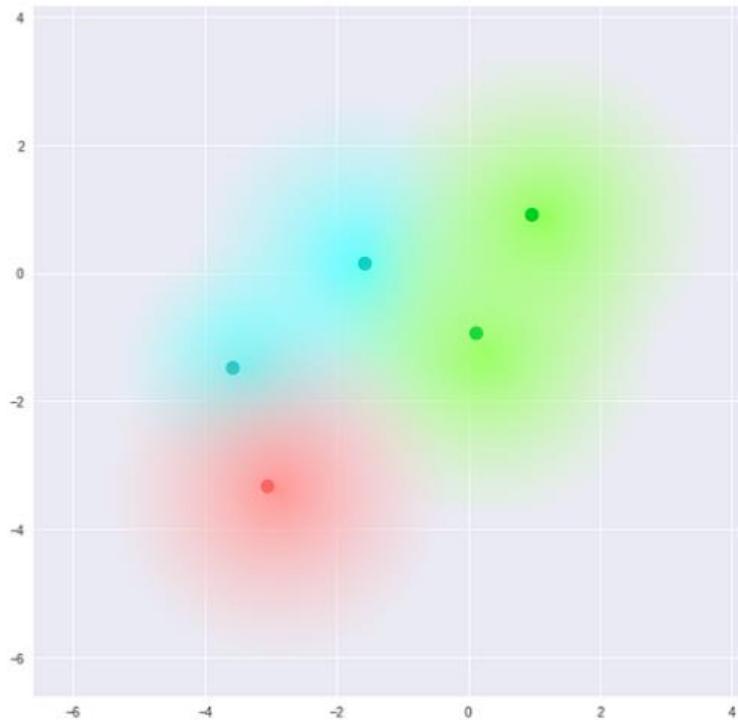
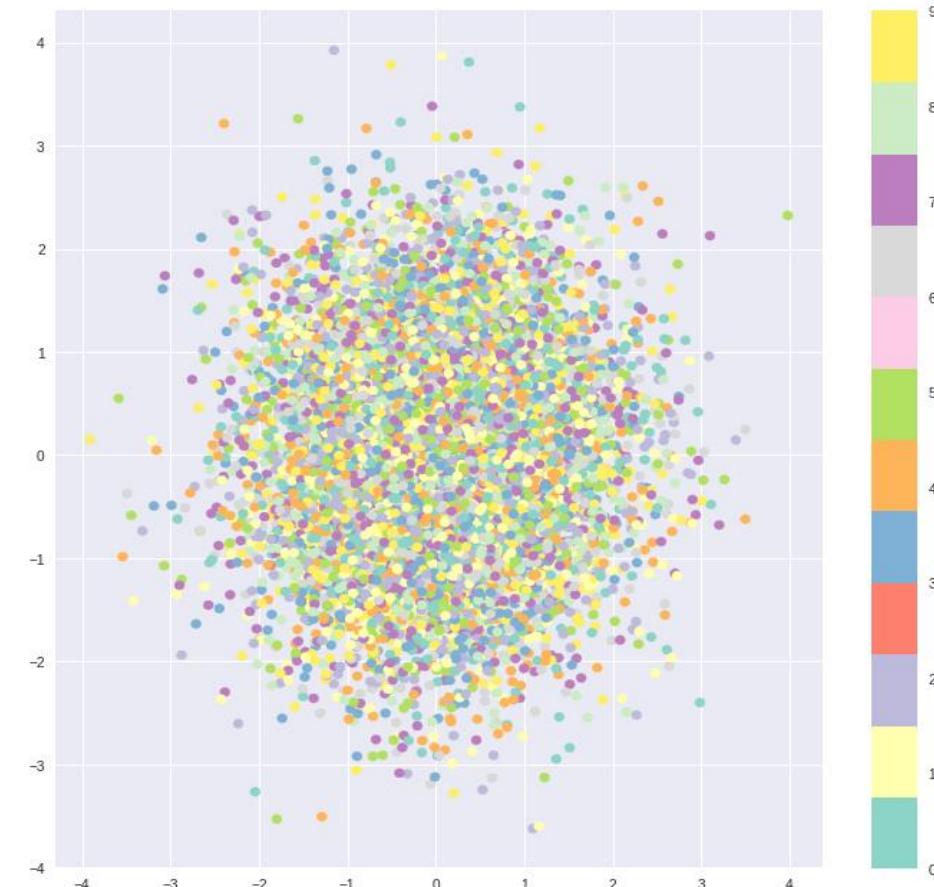


Illustration by Irhum Shafkat

$$l_i(\theta, \phi) = -\mathbb{E}_{z \sim q_\theta(z|x_i)}[\log p_\phi(x_i | z)] + \text{KL}(q_\theta(z | x_i) || p(z))$$

Variational autoencoders

MNIST visualisation
with KL-divergence loss only



Variational autoencoders

MNIST visualisation
with KL-divergence and
classification loss

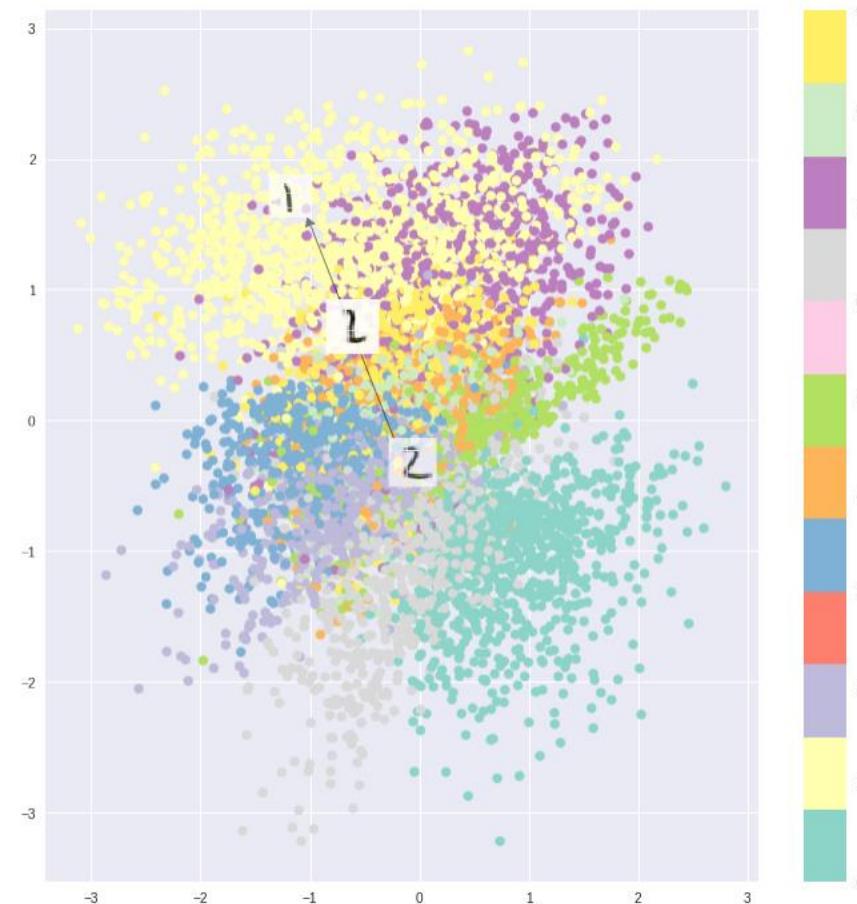


Illustration by Irhum Shafkat

VAE in Keras (encoder)

```
img_shape = (28, 28, 1)
batch_size = 16
latent_dim = 2

input_img = keras.Input(shape=img_shape)

x = layers.Conv2D(32, 3,
                  padding='same', activation='relu')(input_img)
x = layers.Conv2D(64, 3,
                  padding='same', activation='relu',
                  strides=(2, 2))(x)
x = layers.Conv2D(64, 3,
                  padding='same', activation='relu')(x)
x = layers.Conv2D(64, 3,
                  padding='same', activation='relu')(x)
shape_before_flattening = K.int_shape(x)

x = layers.Flatten()(x)
x = layers.Dense(32, activation='relu')(x)

z_mean = layers.Dense(latent_dim)(x)
z_log_var = layers.Dense(latent_dim)(x)
```

Dimensionality of the
latent space: a 2D plane



The input image ends up
being encoded into these
two parameters.

VAE in Keras (sampling)

```
def sampling(args):
    z_mean, z_log_var = args
    epsilon = K.random_normal(shape=(K.shape(z_mean)[0], latent_dim),
                               mean=0., stddev=1.)
    return z_mean + K.exp(z_log_var) * epsilon
z = layers.Lambda(sampling)([z_mean, z_log_var])
```

VAE in Keras (decoder)

```
decoder_input = layers.Input(K.int_shape(z)[1:]) ← Input where you'll feed z
```

```
x = layers.Dense(np.prod(shape_before_flattening[1:]), activation='relu')(decoder_input) | Upsamples the input
```

```
→ x = layers.Reshape(shape_before_flattening[1:])(x)
```

```
x = layers.Conv2DTranspose(32, 3, padding='same', activation='relu', strides=(2, 2))(x)  
x = layers.Conv2D(1, 3, padding='same', activation='sigmoid')(x)
```

Uses a **Conv2DTranspose** layer and **Conv2D** layer to decode z into a feature map the same size as the original image input

```
decoder = Model(decoder_input, x) ←
```

```
z_decoded = decoder(z) ←
```

Applies it to z to recover the decoded z

Instantiates the decoder model, which turns “decoder_input” into the decoded image

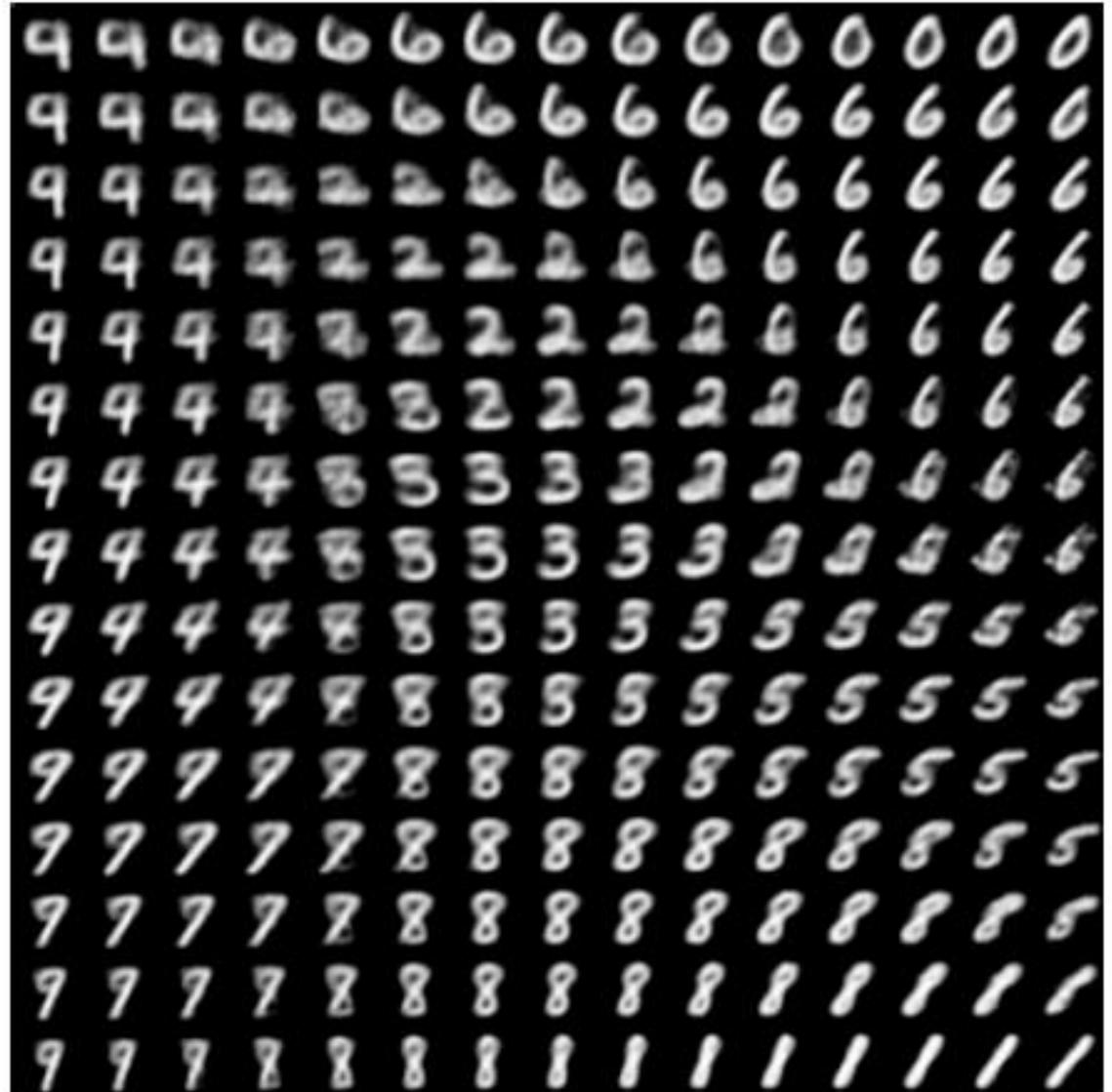
VAE space for MNIST



VAE space for MNIST

The grid of sampled digits

- Specific directions in this space have a meaning: for example, there's a direction for “four-ness,” “one-ness,” and so on.



Questions?