Deep Learning (CS324)

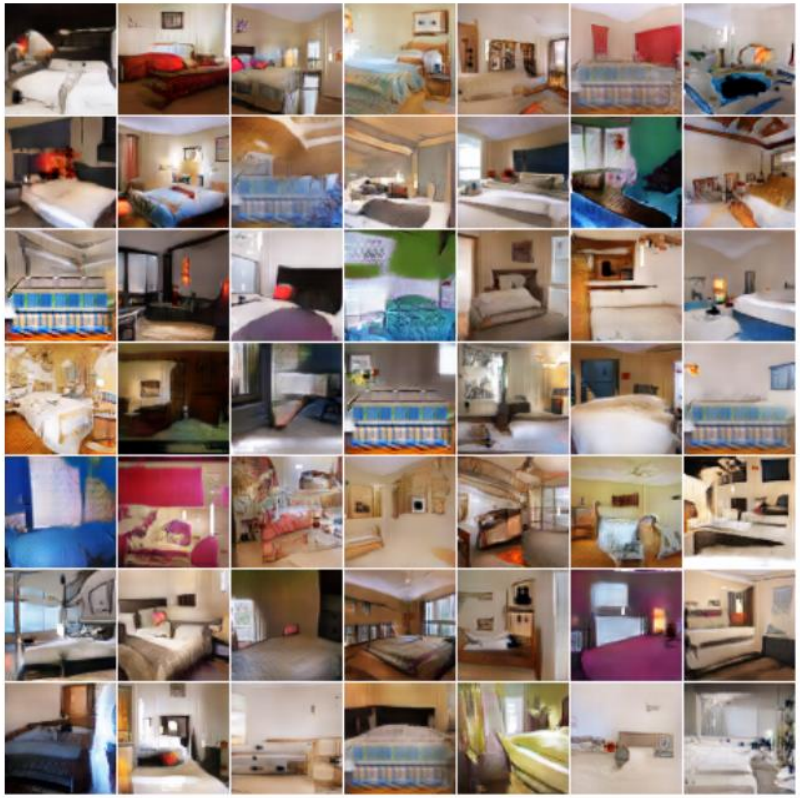
9. Generative adversarial networks*

Prof. Jianguo Zhang
SUSTech

*Based on http://www.deeplearningbook.org chapter 20.10.4 + various sources cited in the slides

Generative tasks

- Generation (from scratch): learn to sample from the distribution represented by the training set
  - *Unsupervised learning* task



Generative tasks

- Conditional generation

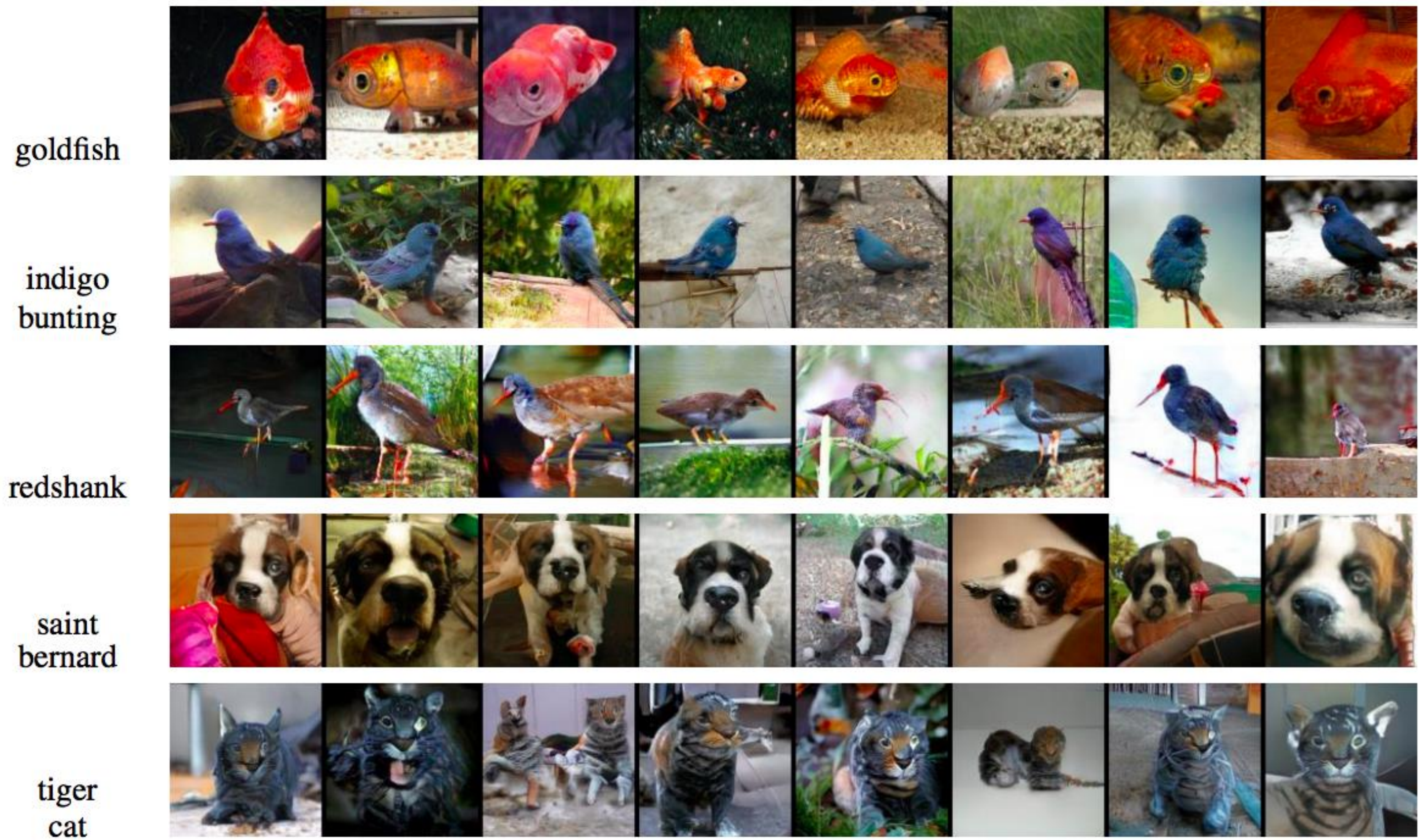
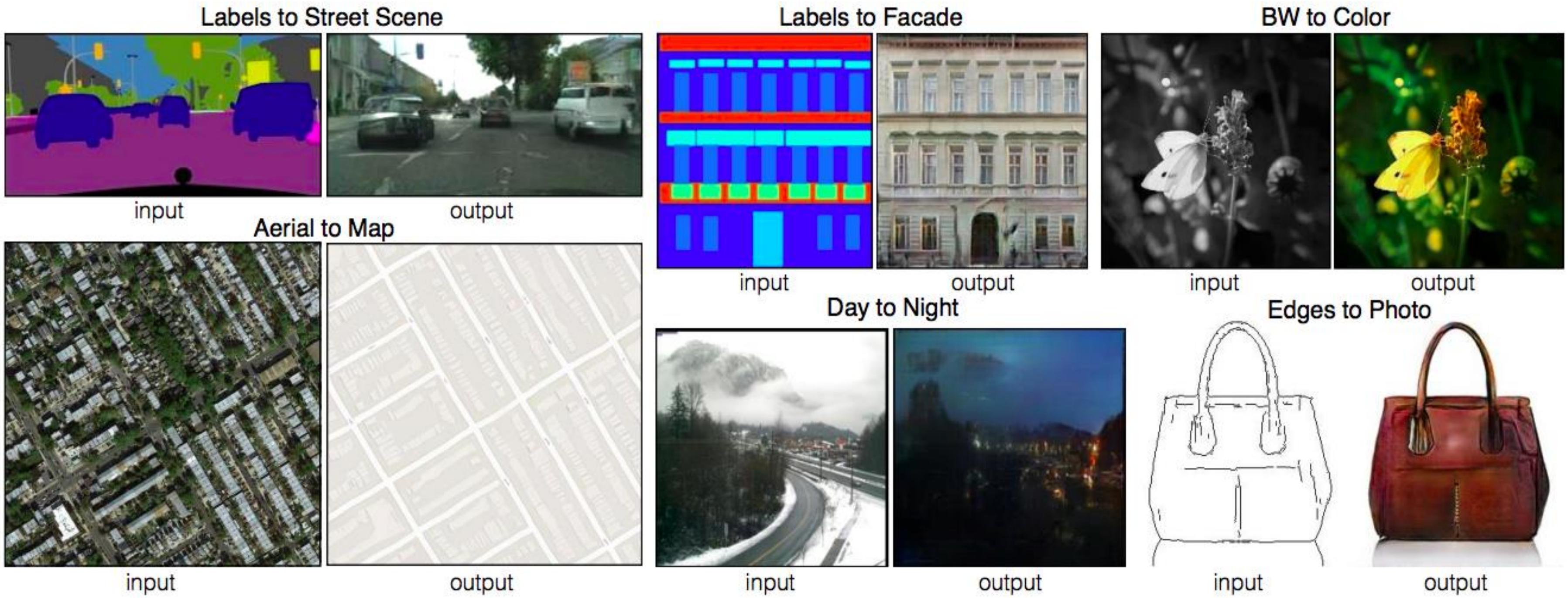


Figure source

Generative tasks

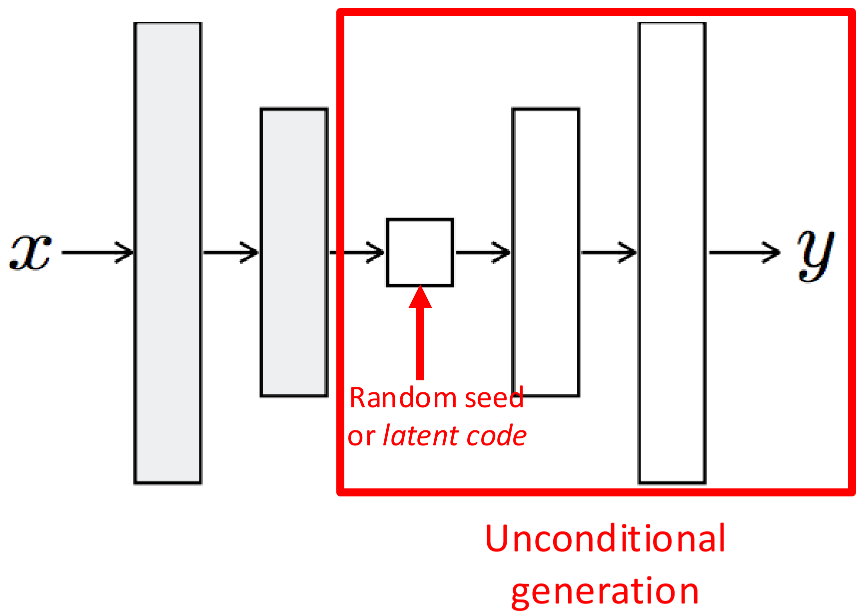
- Image-to-image translation



P. Isola, J.-Y. Zhu, T. Zhou, A. Efros, [Image-to-Image Translation with Conditional Adversarial Networks](), CVPR 2017

Designing a network for generative tasks

- We saw that VAEs can learn to generate data by training both an encoder and a decoder (generator)
- Can we learn just the generator?
  - Recall the decoder of VAE



Learning to sample

Training data $x \sim p_{\text{data}}$

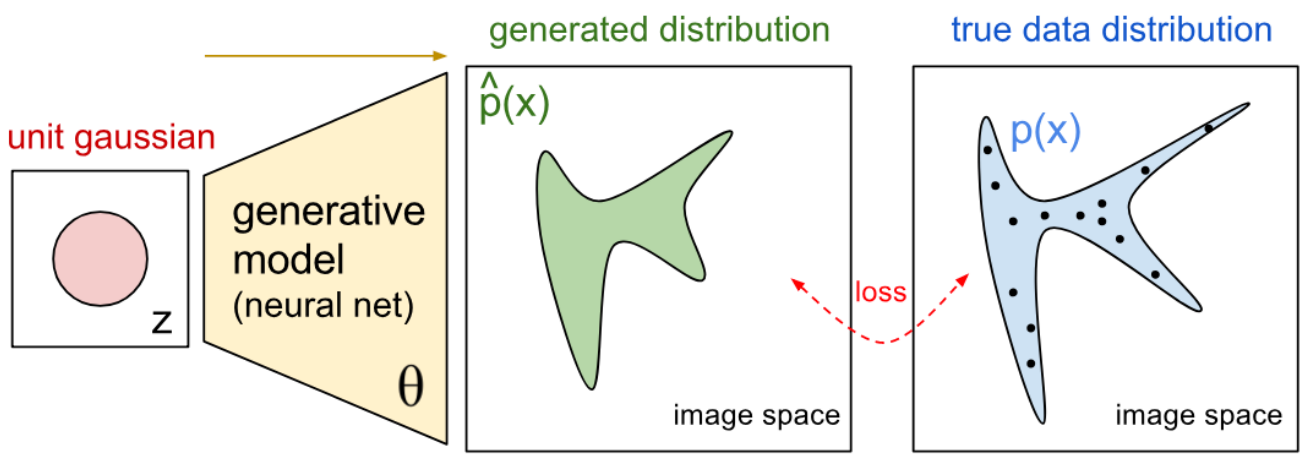
Generated samples $x \sim p_{\text{model}}$

We want to learn p_{model} that matches p_{data}

Adapted from [Stanford CS231n]

From VAEs to GANs

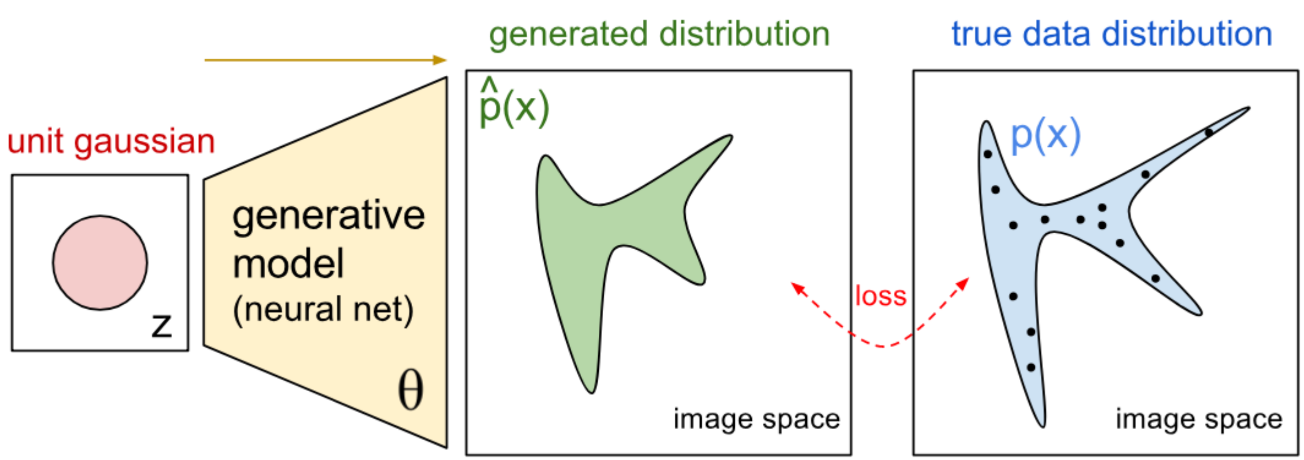
- We saw that VAEs can learn to generate data by training both an encoder and a decoder (generator)
- Can we learn just the generator?



Source: https://openai.com/blog/generative-models/

From VAEs to GANs

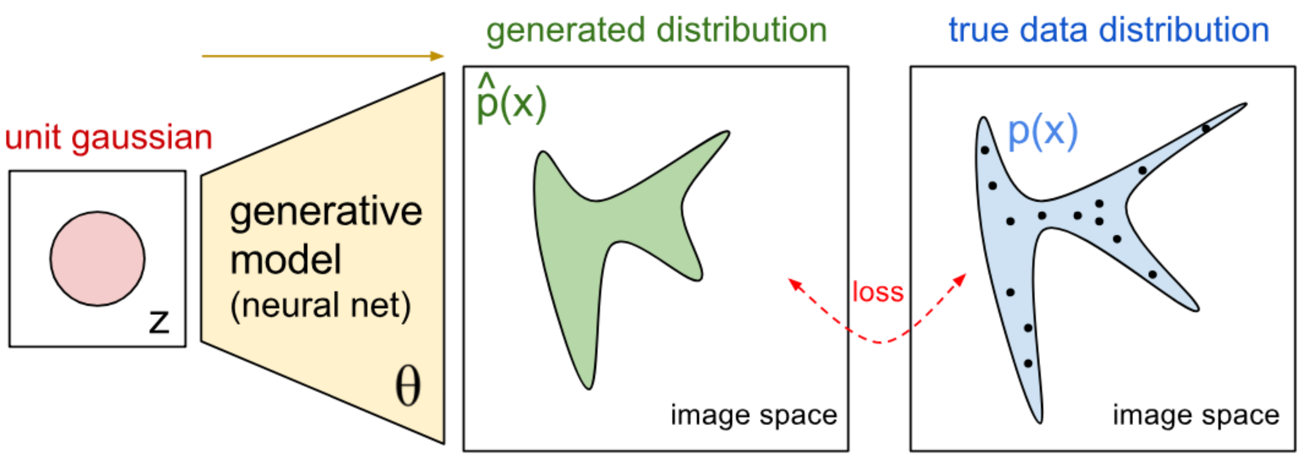
- But, how do we measure the quality of the generator without an encoder?
- In other words, what's the loss function?



Source: https://openai.com/blog/generative-models/

Generative adversarial networks

- GANs propose to learn the loss function
- The training process is a game between two networks



Source: https://openai.com/blog/generative-models/

Generative adversarial networks

- Train two networks with opposing objectives:
  - **Generator:** learns to generate samples
  - **Discriminator:** learns to distinguish between generated and real samples

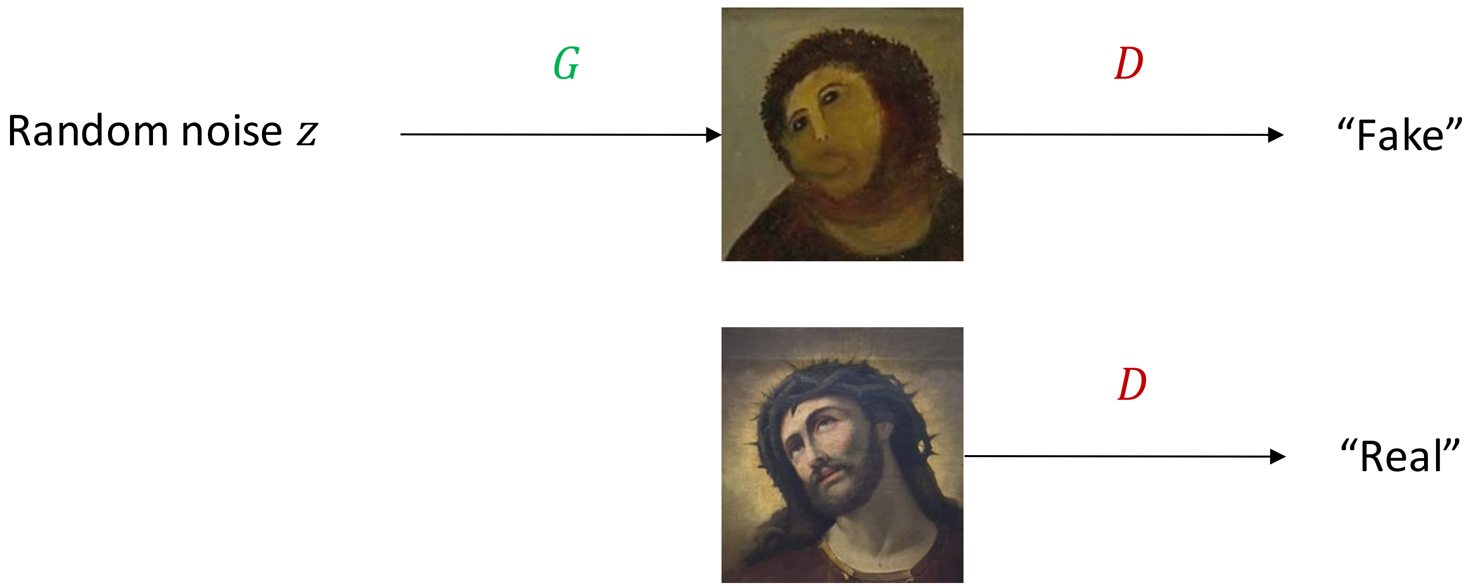
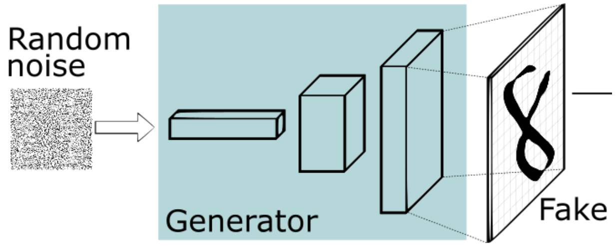


Figure adapted from
F. Fleuret

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets, NIPS 2014

Generative adversarial networks

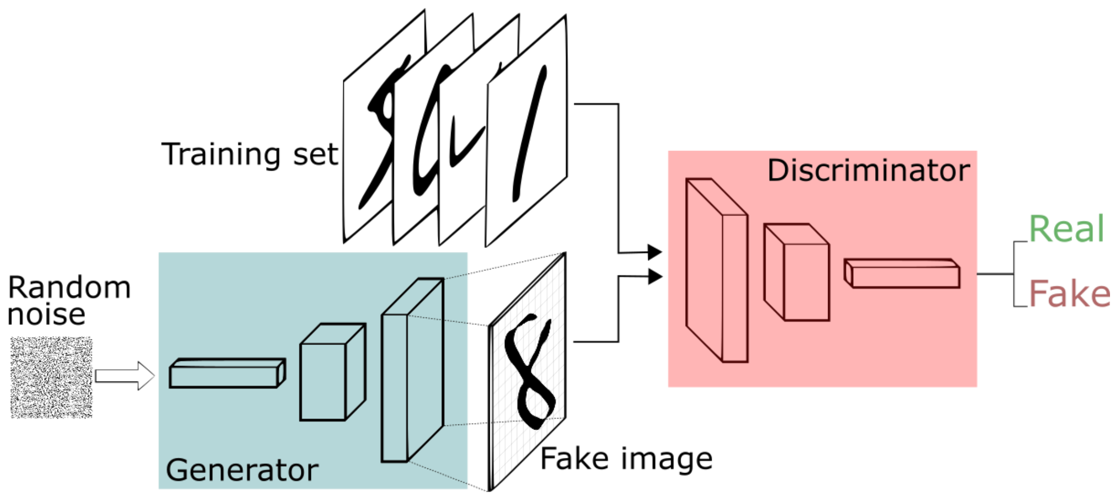
- **Generator**: G(z) takes random noise z as input and outputs a (fake) image



Source: https://medium.freecodecamp.org/an-intuitive-introduction-to-generative-adversarial-networks-gans-7a2264a81394

Generative adversarial networks

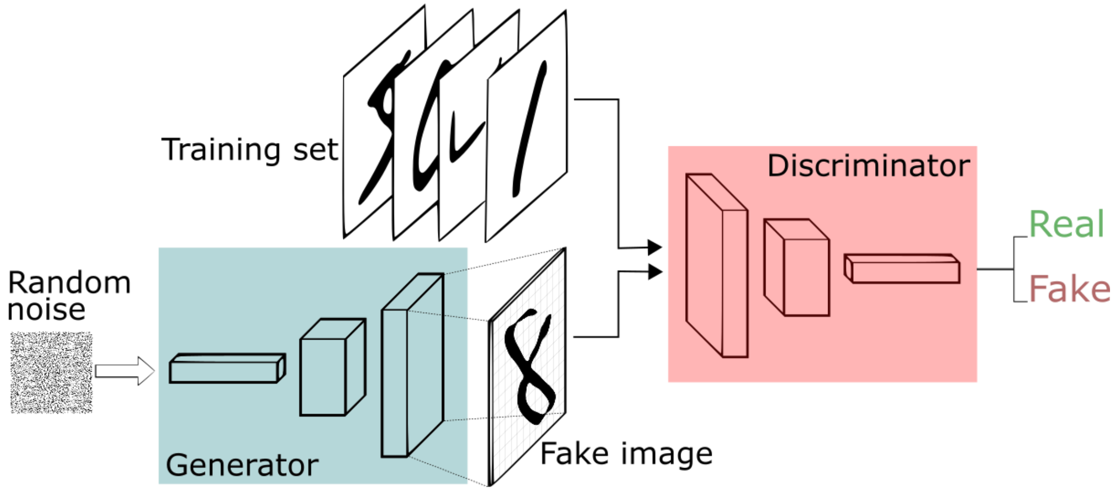
- **Generator**: G(*z*) takes random noise *z* as input and outputs a (fake) image
- **Discriminator**: D(*x*) receives an image x in input, real or fake, and estimates its probability to be real



Source: https://medium.freecodecamp.org/an-intuitive-introduction-to-generative-adversarial-networks-gans-7a2264a81394

Generative adversarial networks

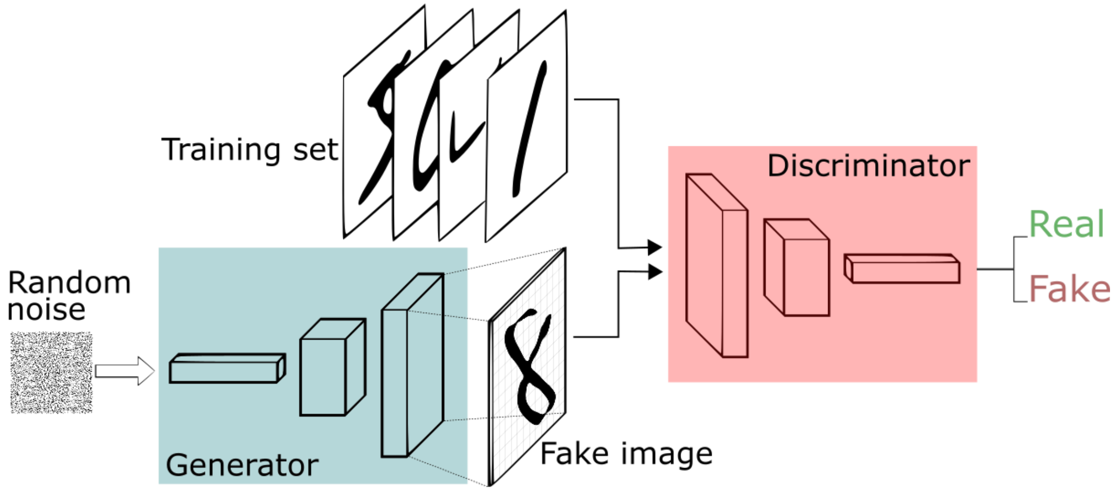
- **Adversarial training**: the generator tries to fool the discriminator while the discriminator tries to get better at distinguishing fake vs real images



Source: https://medium.freecodecamp.org/an-intuitive-introduction-to-generative-adversarial-networks-gans-7a2264a81394

Generative adversarial networks

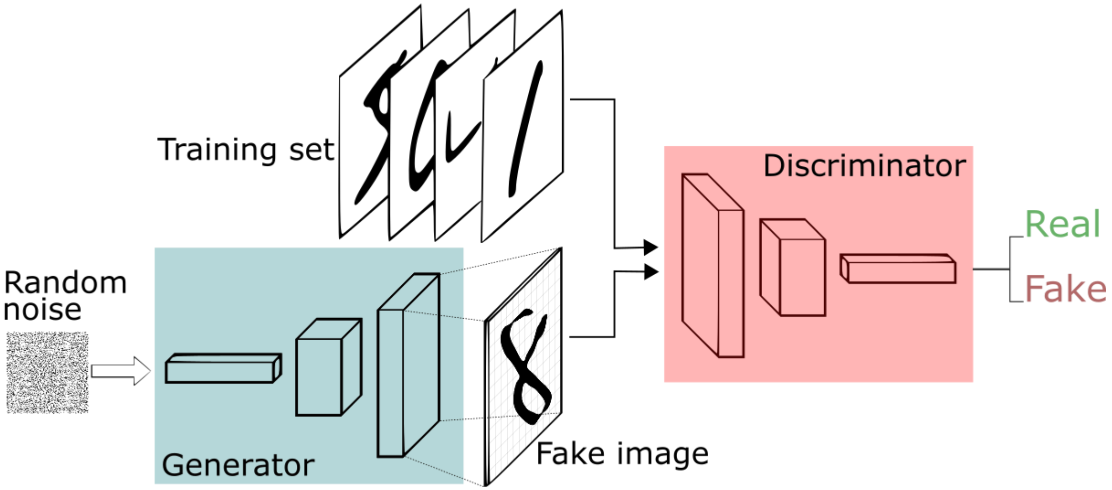
- When the discriminator spots a fake the generator adjusts its parameters, until at the end the generator reproduces the true data distribution and the discriminator is unable to find differences



Source: https://medium.freecodecamp.org/an-intuitive-introduction-to-generative-adversarial-networks-gans-7a2264a81394

Generative adversarial networks

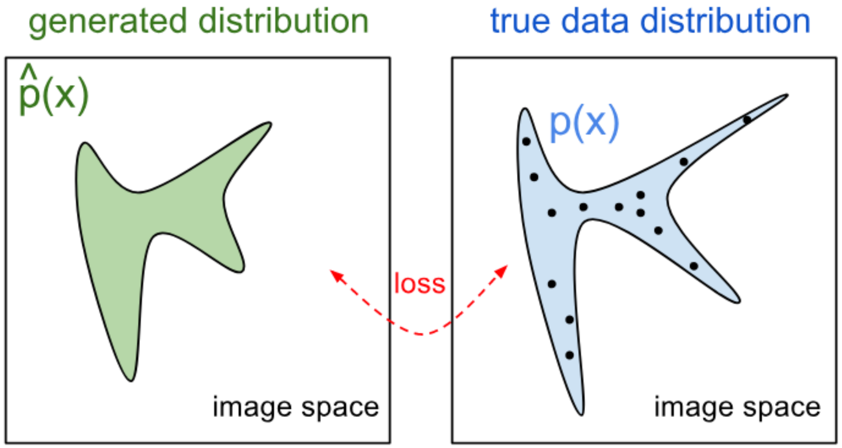
- **Note**: both the generator and the discriminator need to be <u>differentiable</u>
- Typically both implemented as (deep) neural **networks**, so we can use backpropagation



Source: https://medium.freecodecamp.org/an-intuitive-introduction-to-generative-adversarial-networks-gans-7a2264a81394

Generative adversarial networks

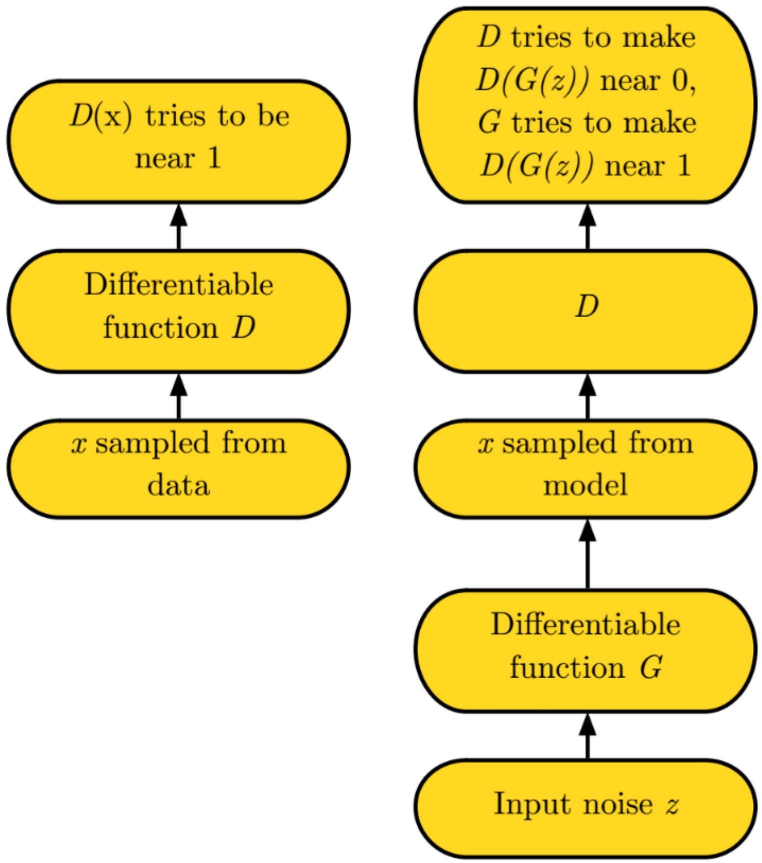
- That’s like saying that the discriminator at the end in unable to tell the difference between the generated data distribution and the true data distribution (where training data comes from)



Source: https://medium.freecodecamp.org/an-intuitive-introduction-to-generative-adversarial-networks-gans-7a2264a81394

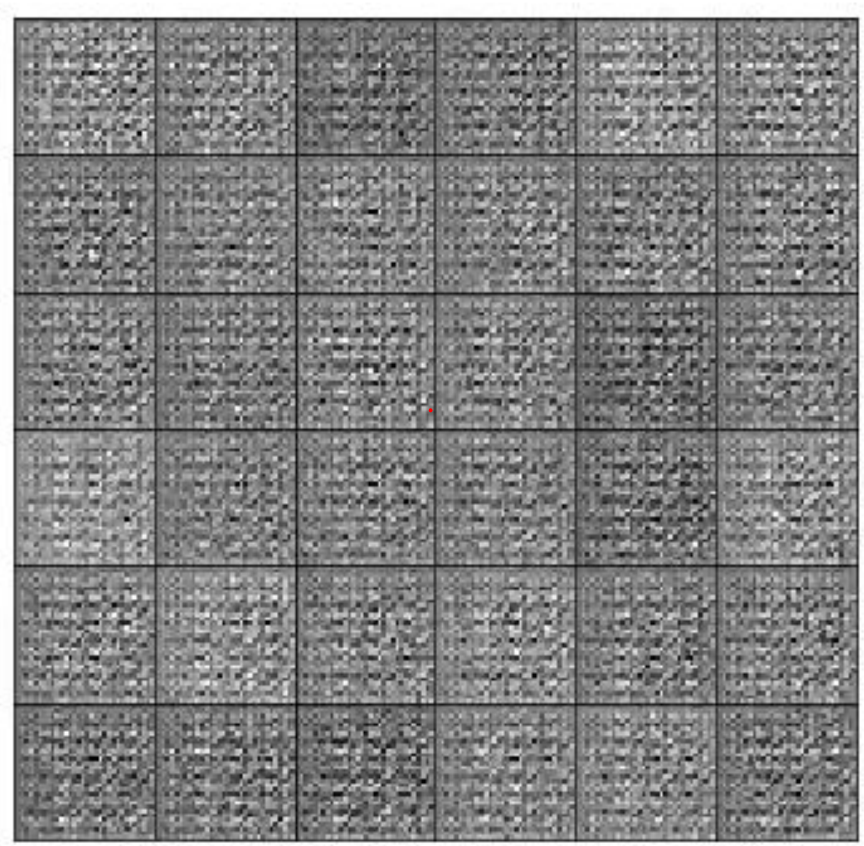
GANs pipeline

Real



Fake

Source: UVA 2018 deep learning course



Source: https://medium.freecodecamp.org/an-intuitive-introduction-to-generative-adversarial-networks-gans-7a2264a81394

Real

Fake

Source: https://medium.freecodecamp.org/an-intuitive-introduction-to-generative-adversarial-networks-gans-7a2264a81394

Minimax and zero-sum games

- G and D follow a **minimax strategy**

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Minimax and zero-sum games

- G and D follow a **minimax strategy**

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Prior distribution on input noise variables

Minimax and zero-sum games

- G and D follow a **minimax strategy**

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

The generator is a differentiable function (e.g., MLP) mapping noise to data space

Minimax and zero-sum games

- G and D follow a **minimax strategy**

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

The discriminator is a differentiable function (e.g., MLP) with single scalar output, i.e., the probability that *x* comes from the true data distribution

Minimax and zero-sum games

- G and D follow a **minimax strategy**

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

D(*x*) = 1 when the discriminator thinks *x* is a real image

Minimax and zero-sum games

- G and D follow a **minimax strategy**

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

D(G(z)) = 1 when the discriminator thinks G(z) is a real image

Minimax and zero-sum games

- G and D follow a **minimax strategy**

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

D is trained to maximise the probability of giving the correct label to samples from the true data distribution (the label should be 1) as well as to samples from G (the label should be 0)

Minimax and zero-sum games

- G and D follow a **minimax strategy**

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Large when D(*x*) close to 1

Minimax and zero-sum games

- G and D follow a **minimax strategy**

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Large when D(G(*z*)) close to 0

Minimax and zero-sum games

- G and D follow a **minimax strategy**

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Simultaneously G is trained to minimise log(1 − D(G(*z*))), i.e., fool the discriminator

Minimax and zero-sum games

- G and D follow a **minimax strategy**

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

- For a fixed G, the loss is effectively the binary cross-entropy
- The minimax strategy is used for **zero-sum games**, i.e., loss of one player = gain of the adversary
- The minimax solution is the **Nash equilibrium**

Nash equilibrium is a state in which D and G don't have any incentive to change because doing so would make the value of the objective function worse

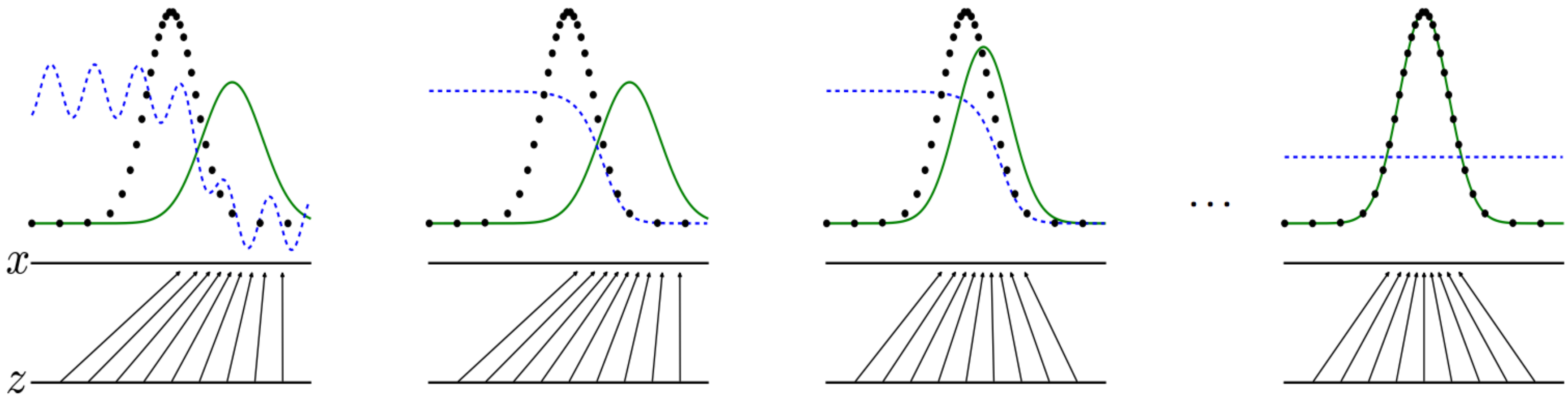
Minimax and zero-sum games

- G and D follow a **minimax strategy**

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

- For a fixed G, the loss is effectively the binary cross-entropy
- The minimax strategy is used for **zero-sum games**, i.e., loss of one player = gain of the adversary
- The minimax solution is the **Nash equilibrium**

In machine learning terms it means we reached convergence of the gradient descent for the joint optimisation problem



(a) (b) (c) (d)

Figure 1: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which $\boldsymbol{z}$ is sampled, in this case uniformly. The horizontal line above is part of the domain of $\boldsymbol{x}$. The upward arrows show how the mapping $\boldsymbol{x} = G(\boldsymbol{z})$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g. (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$. (c) After an update to G, gradient of D has guided $G(\boldsymbol{z})$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{\text{data}}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(\boldsymbol{x}) = \frac{1}{2}$.

Source: https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

GAN training

- $V(G, D) = \mathbb{E}_{x \sim p_{\text{data}}} \log D(x) + \mathbb{E}_{z \sim p} \log(1 - D(G(z)))$

- Alternate between
  - *Gradient ascent* on discriminator:

$$D^* = \arg\max_D V(G, D)$$

  - *Gradient descent* on generator (minimize log-probability of discriminator being right):

$$\begin{aligned} G^* &= \arg\min_G V(G, D) \\ &= \arg\min_G \mathbb{E}_{z \sim p} \log(1 - D(G(z))) \end{aligned}$$

  - In practice, do *gradient ascent* on generator (maximize log-probability of discriminator being wrong):

$$G^* = \arg\max_G \mathbb{E}_{z \sim p} \log(D(G(z)))$$

Training GANs

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 $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ 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 $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ 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.

Source: https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

Training GANs

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

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

In practice this saturates in the beginning since D rejects the samples with high confidence because they are clearly different from the training data

end for

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

Source: https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

Training GANs

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 $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.

  • Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ 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 $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.

  • Update the generator by descending its stochastic gradient:

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

So we use this loss function for the generator instead, i.e., the generator maximises the log-prob of the discriminator being mistaken

end for

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

Source: https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

GAN training

$\min_{w_G} \mathbb{E}_{z\sim p} \log(1 - D(G(z)))$ vs. $\max_{w_G} \mathbb{E}_{z\sim p} \log(D(G(z)))$

Large gradients for low-quality samples

Small gradients for high-quality samples

Low discriminator score (low-quality samples)

High discriminator score (high-quality samples)

Want to learn from confidently rejected sample but gradients here are small

These samples already fool the discriminator so we don't need large gradients here

Figure source

Example generator: DCGAN

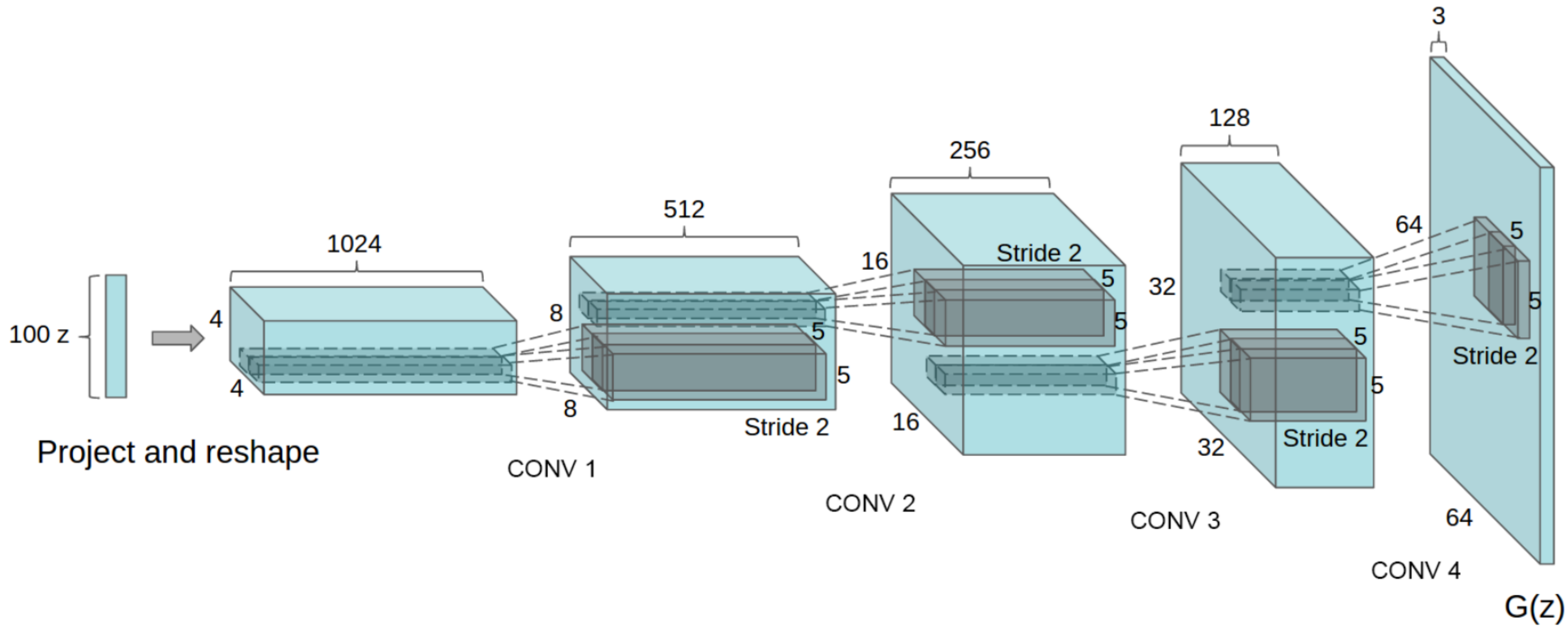
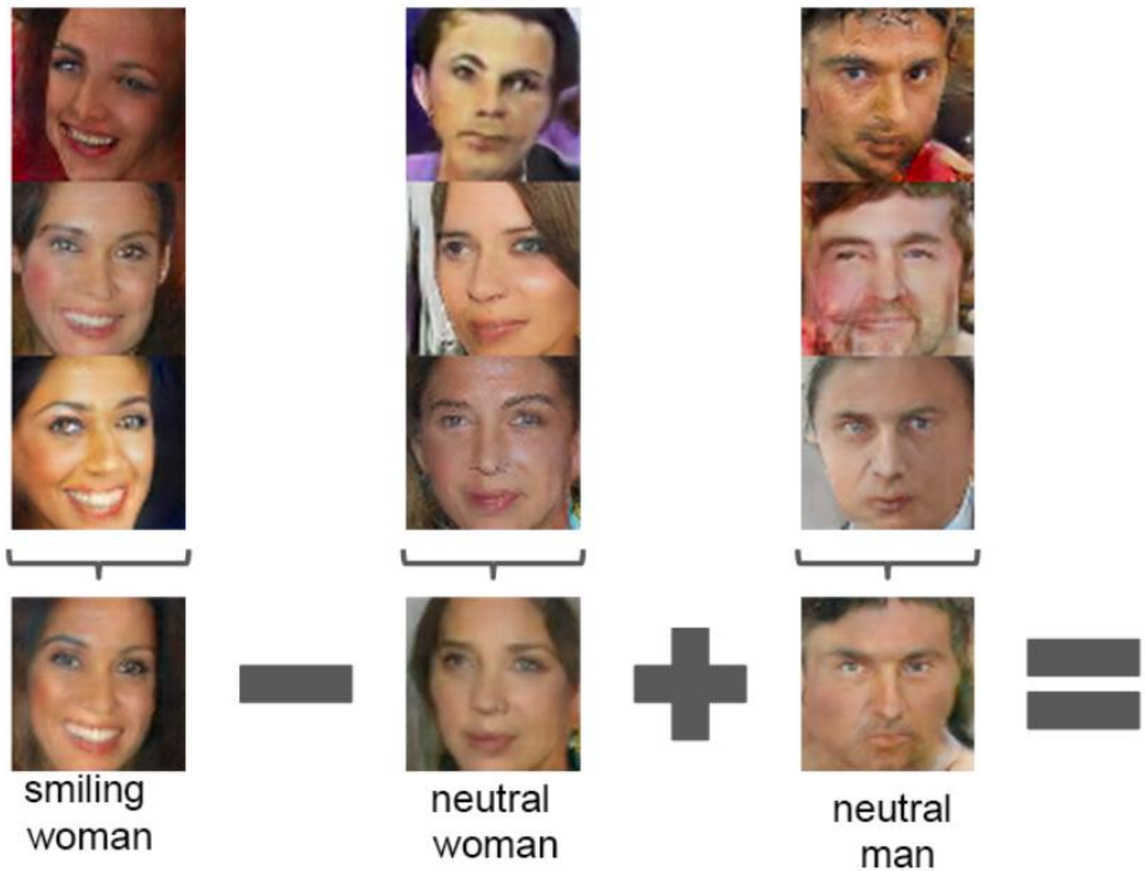


Figure 1: DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions) then convert this high level representation into a 64×64 pixel image. Notably, no fully connected or pooling layers are used.

Source: https://arxiv.org/pdf/1511.06434.pdf

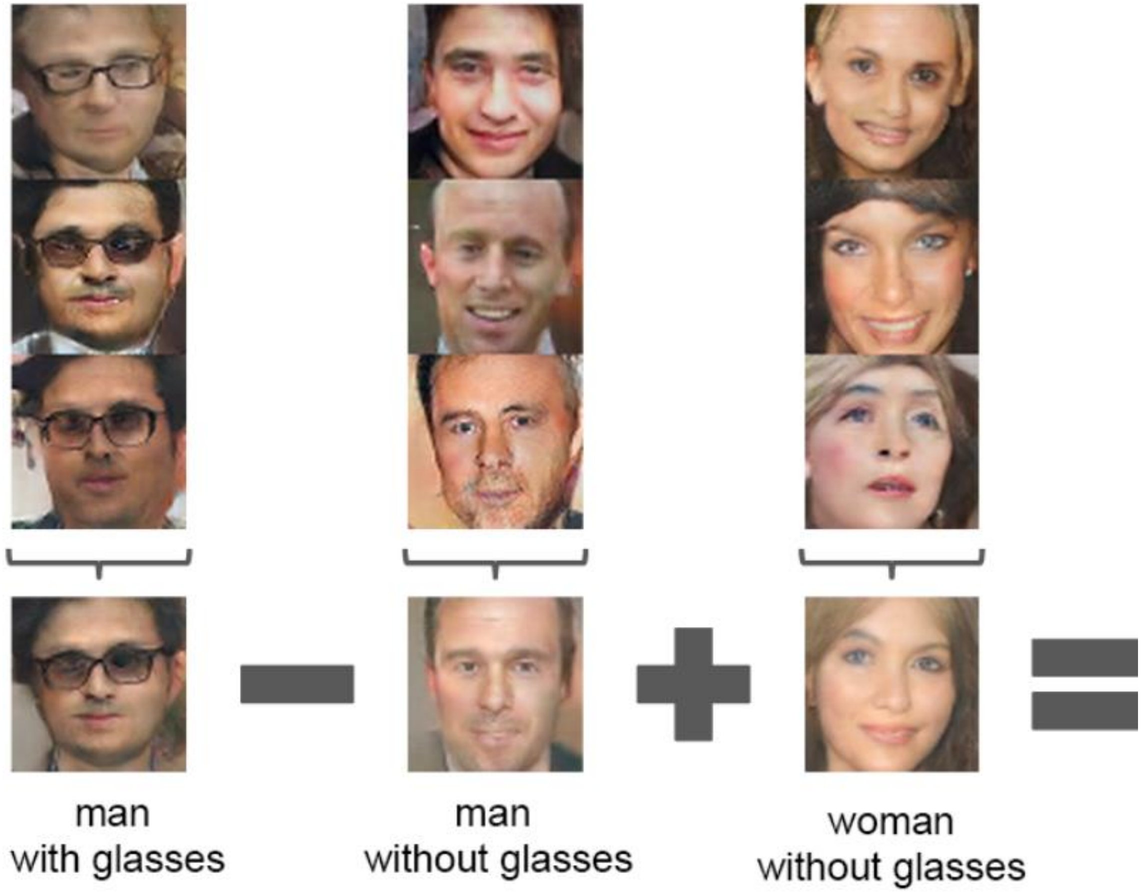
DCGAN results

- Vector arithmetic in the z space



DCGAN results

- Vector arithmetic in the z space



DCGAN results

- Pose transformation by adding a “turn” vector

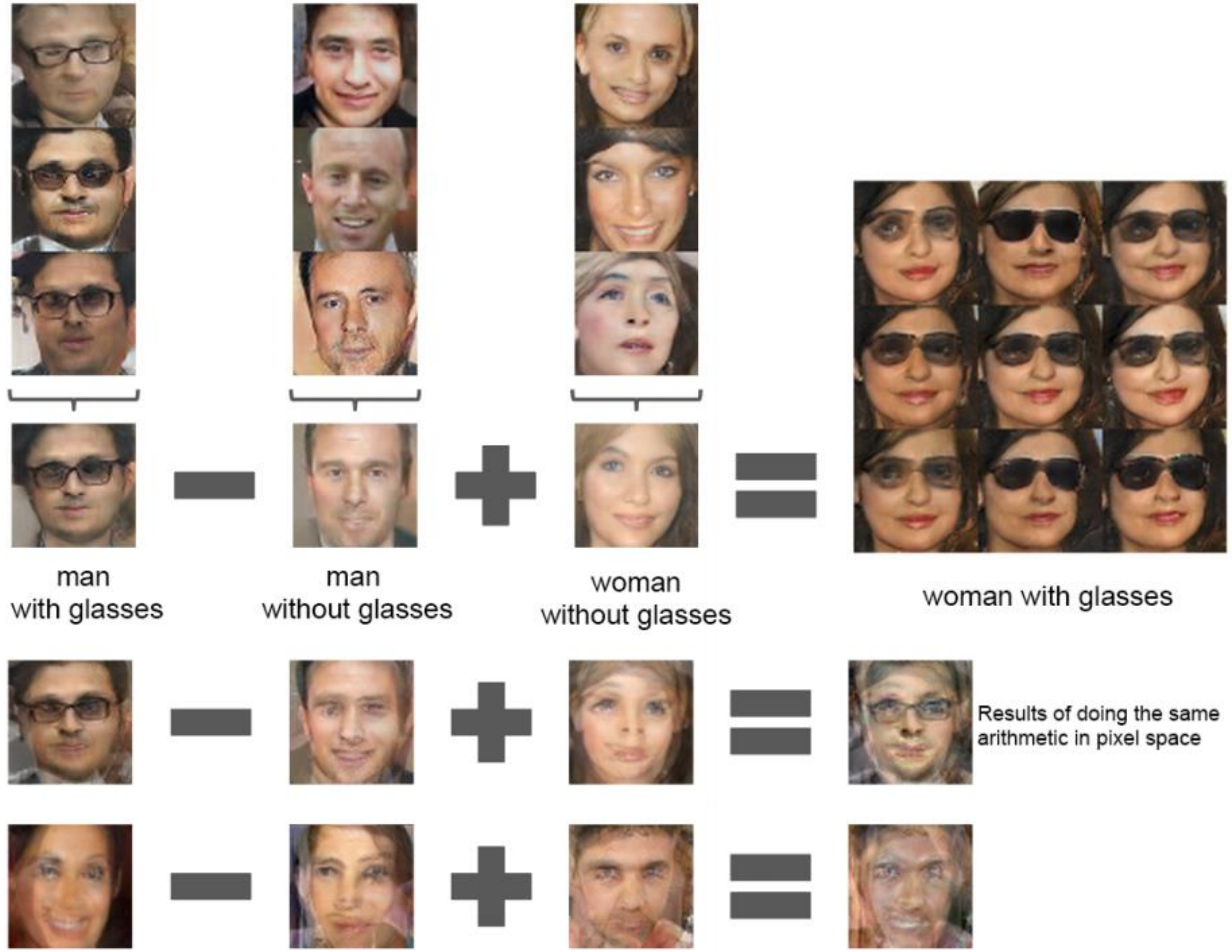


Figure 7: Vector arithmetic for visual concepts. For each column, the Z vectors of samples are averaged. Arithmetic was then performed on the mean vectors creating a new vector Y. The center sample on the right hand side is produce by feeding Y as input to the generator. To demonstrate the interpolation capabilities of the generator, uniform noise sampled with scale +-0.25 was added to Y to produce the 8 other samples. Applying arithmetic in the input space (bottom two examples) results in noisy overlap due to misalignment.

Source: https://arxiv.org/pdf/1511.06434.pdf

Problems with GANs

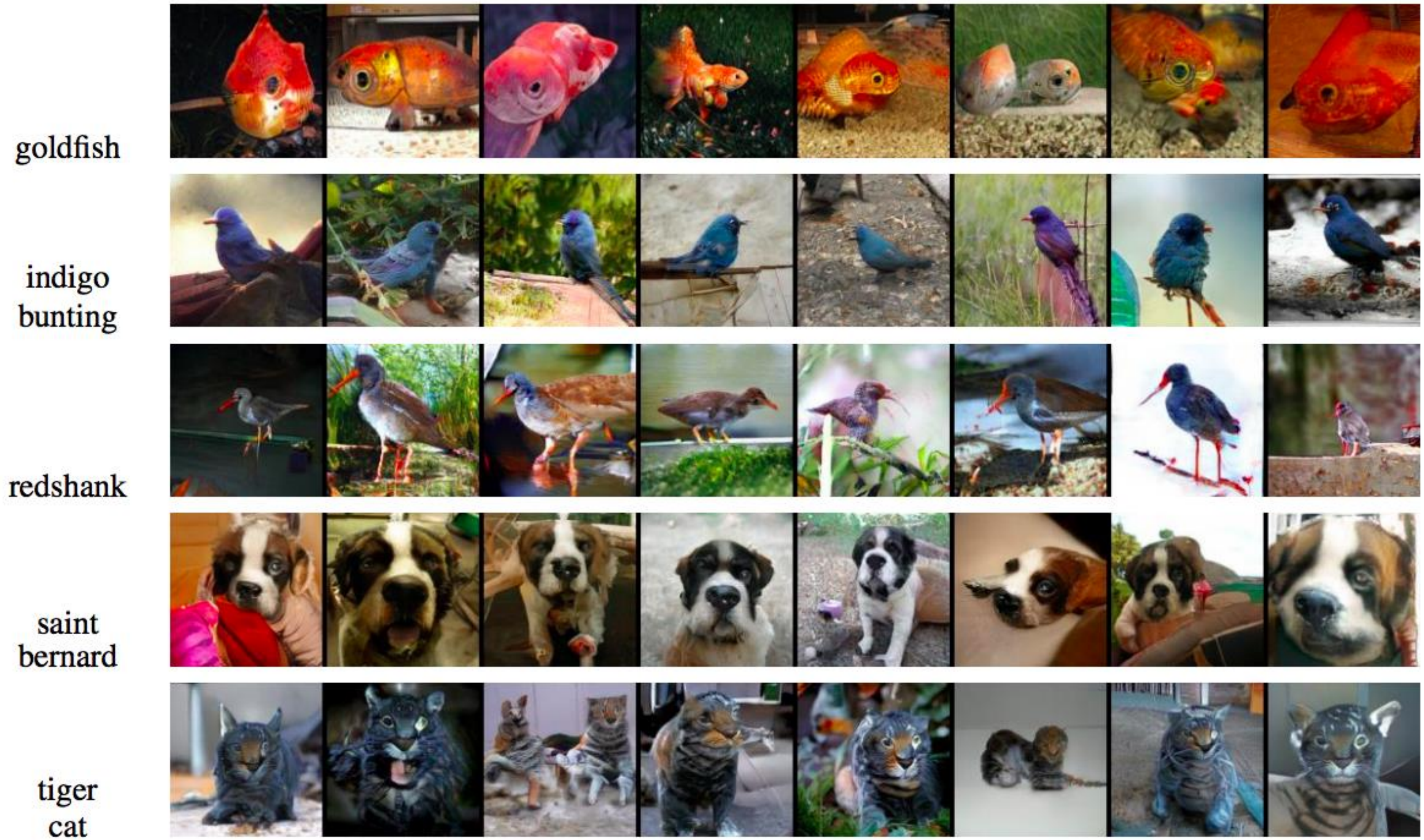
- In the real world we face multiple problems when training GANs:
  - Finite sample size: training set is finite, not the full distribution
  - Limited capacity: the generator has limit capacity, i.e. cannot perfectly represent any distribution
  - Optimisation errors: optimisers can get stuck in local optima or never exactly converge to global optima

Problems with GANs

- In the real world we face multiple problems when training GANs:
  - Saddle point problem: harder than finding a maximum or minimum
  - Balancing updates: D too weak/strong means no gradient for G to improve

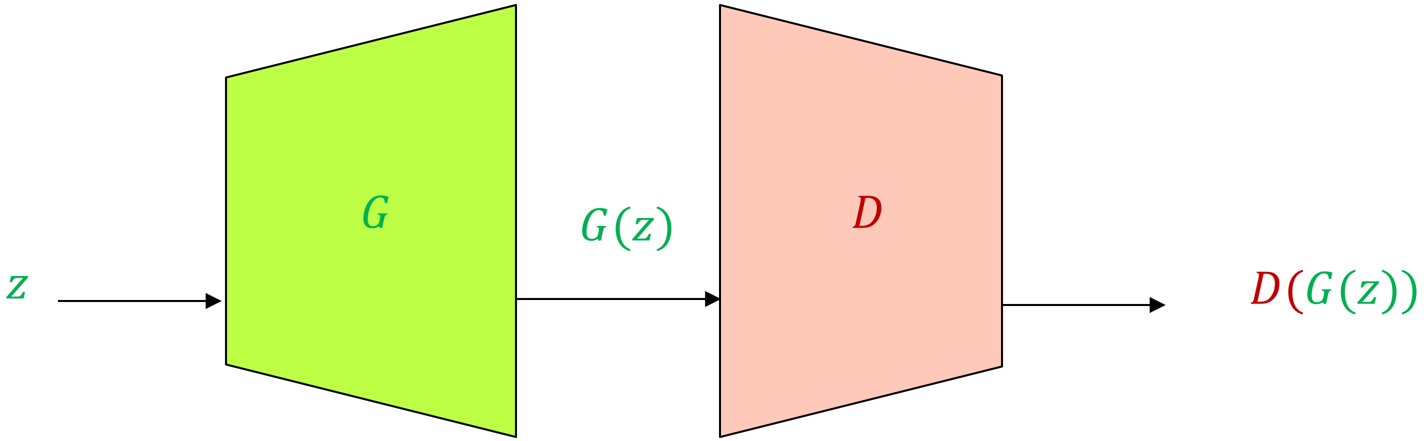
M. Arjovsky, S. Chintala, L. Bottou, Wasserstein generative adversarial networks

Conditional generation (cGAN)



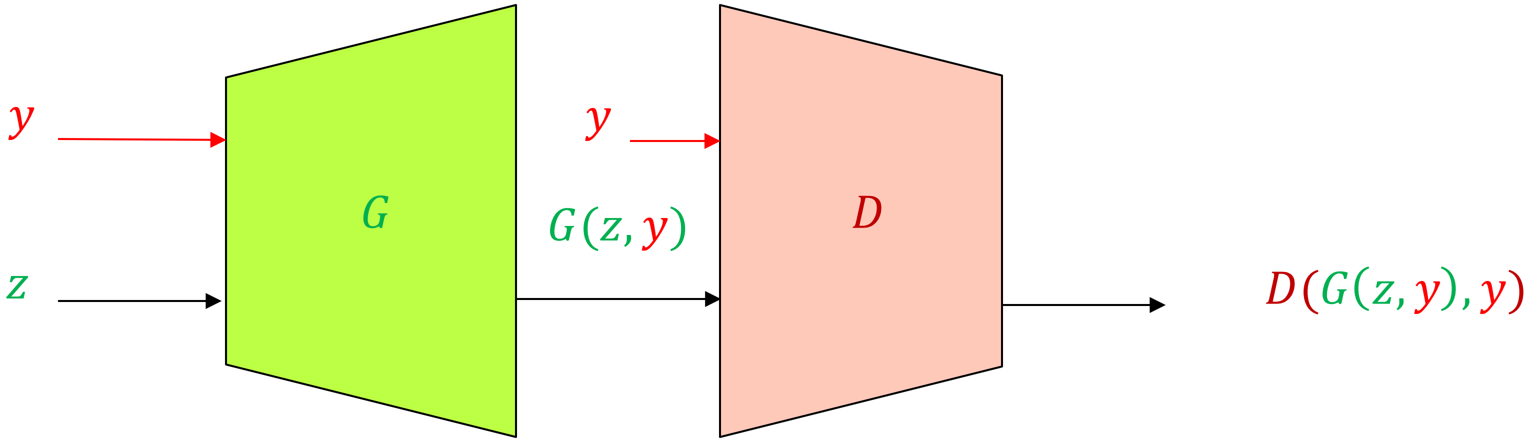
Conditional generation (cGAN)

- Suppose we want to condition the generation of samples on discrete side information (label) y
  - How do we add y to the basic GAN framework?



Conditional generation

- Suppose we want to condition the generation of samples on discrete side information (label) y
  - How do we add y to the basic GAN framework?



Conditional generation

- Example: simple network for generating 28 x 28 MNIST digits

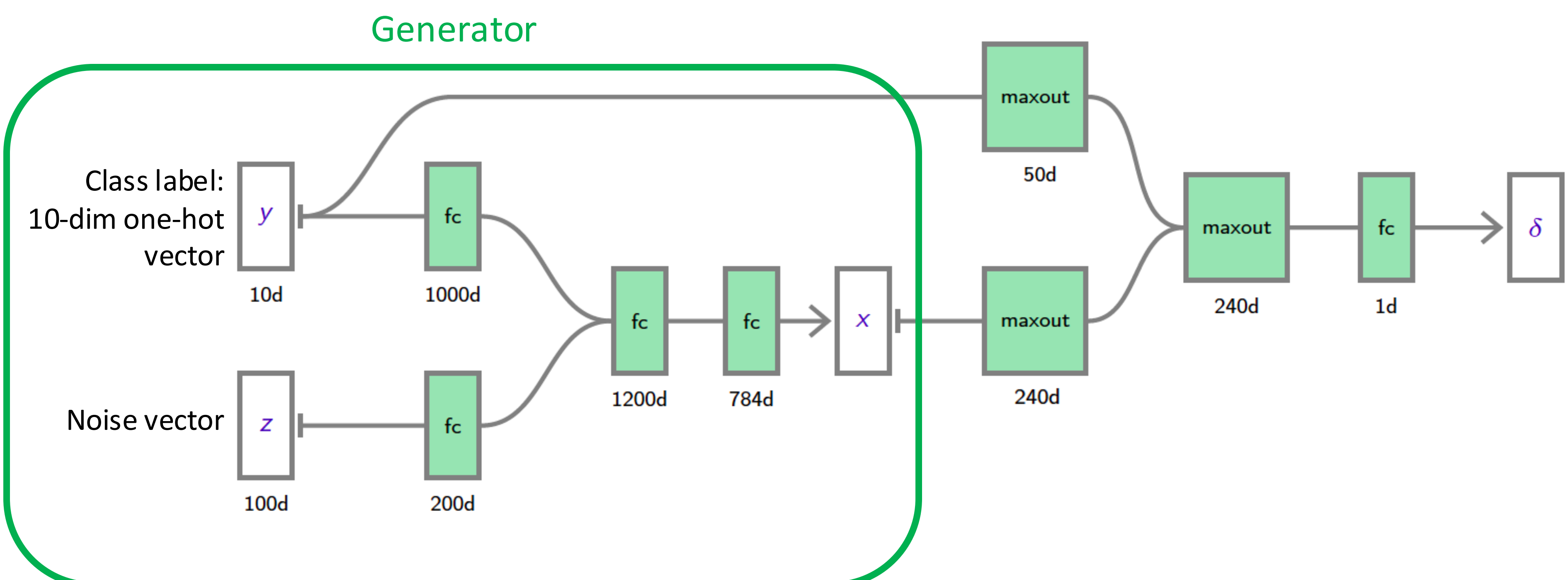


Figure source: [F. Fleuret]()

M. Mirza and S. Osindero, [Conditional Generative Adversarial Nets](), arXiv 2014

Conditional generation

- Example: simple network for generating 28 x 28 MNIST digits

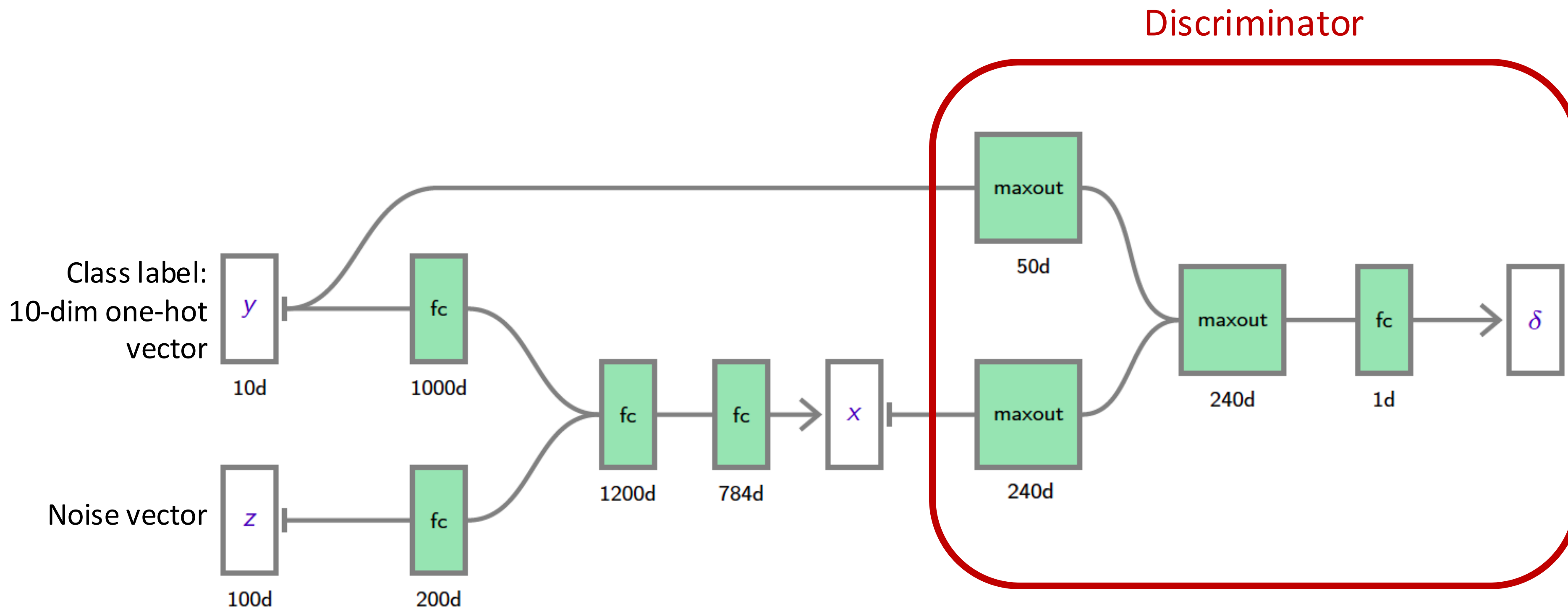
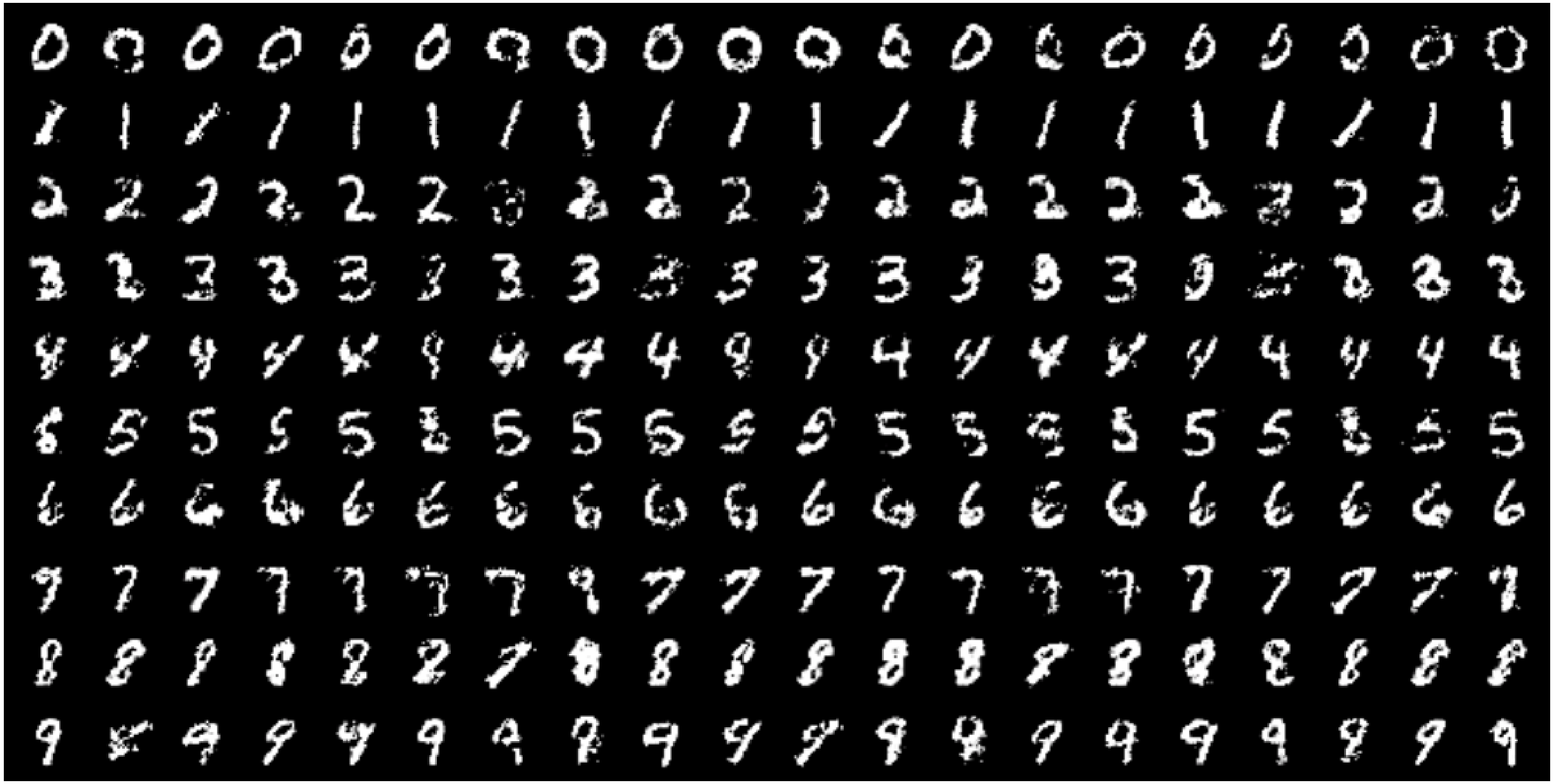


Figure source: F. Fleuret

M. Mirza and S. Osindero, Conditional Generative Adversarial Nets, arXiv 2014

Conditional generation

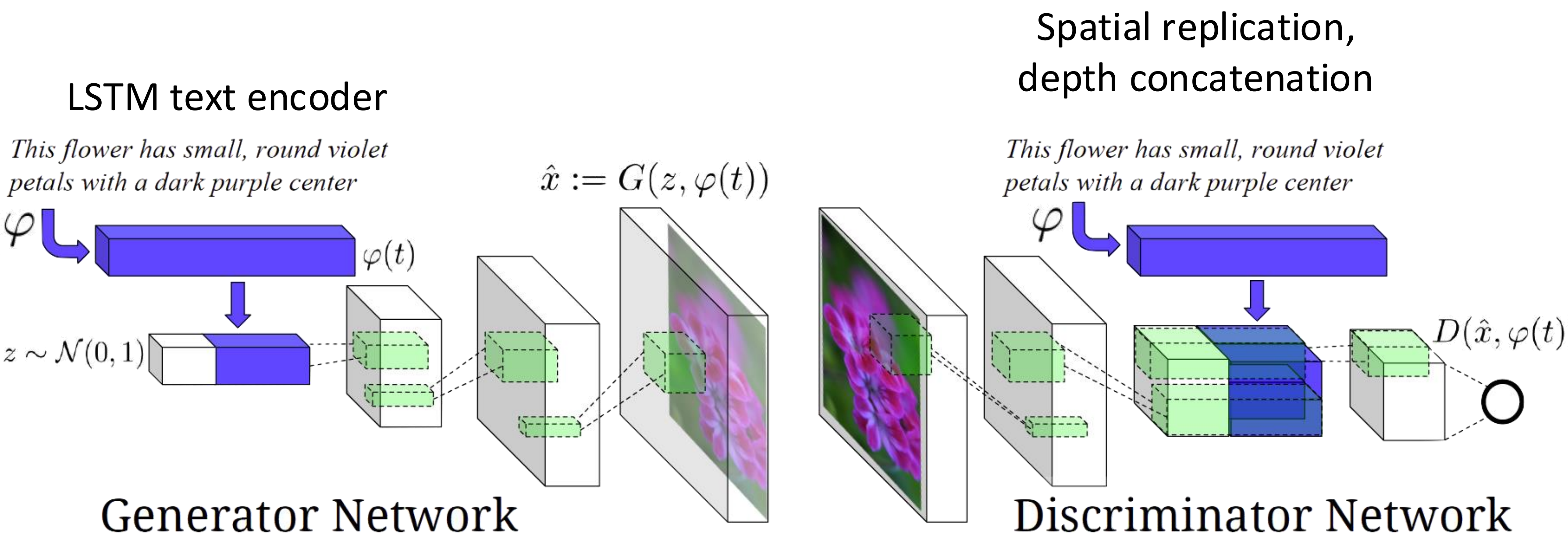
- Example: simple network for generating 28 x 28 MNIST digits



M. Mirza and S. Osindero, [Conditional Generative Adversarial Nets](), arXiv 2014

Conditional generation

- Another example: text-to-image synthesis

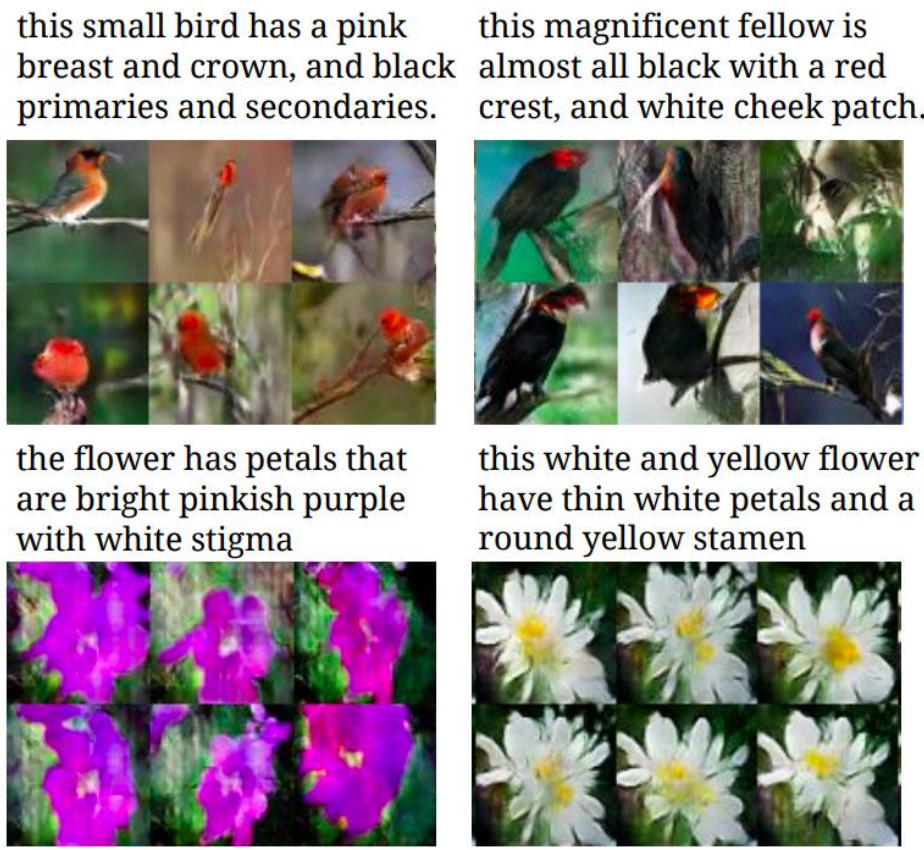


S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, H. Lee, [Generative adversarial text to image synthesis](), ICML 2016

Conditional generation

- Another example: text-to-image synthesis

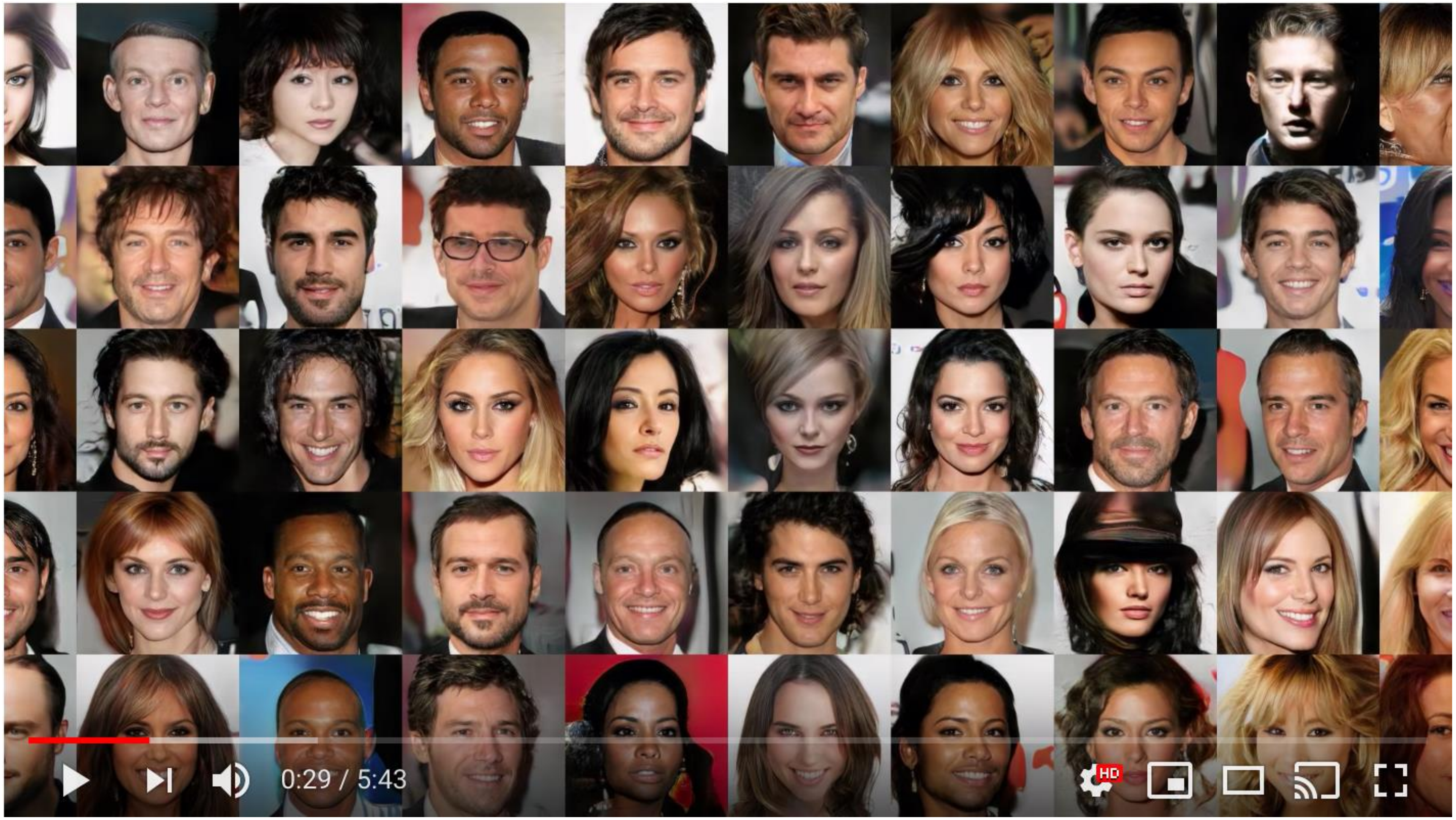
Previously unseen captions (*zero-shot* setting)



Captions seen in the training set

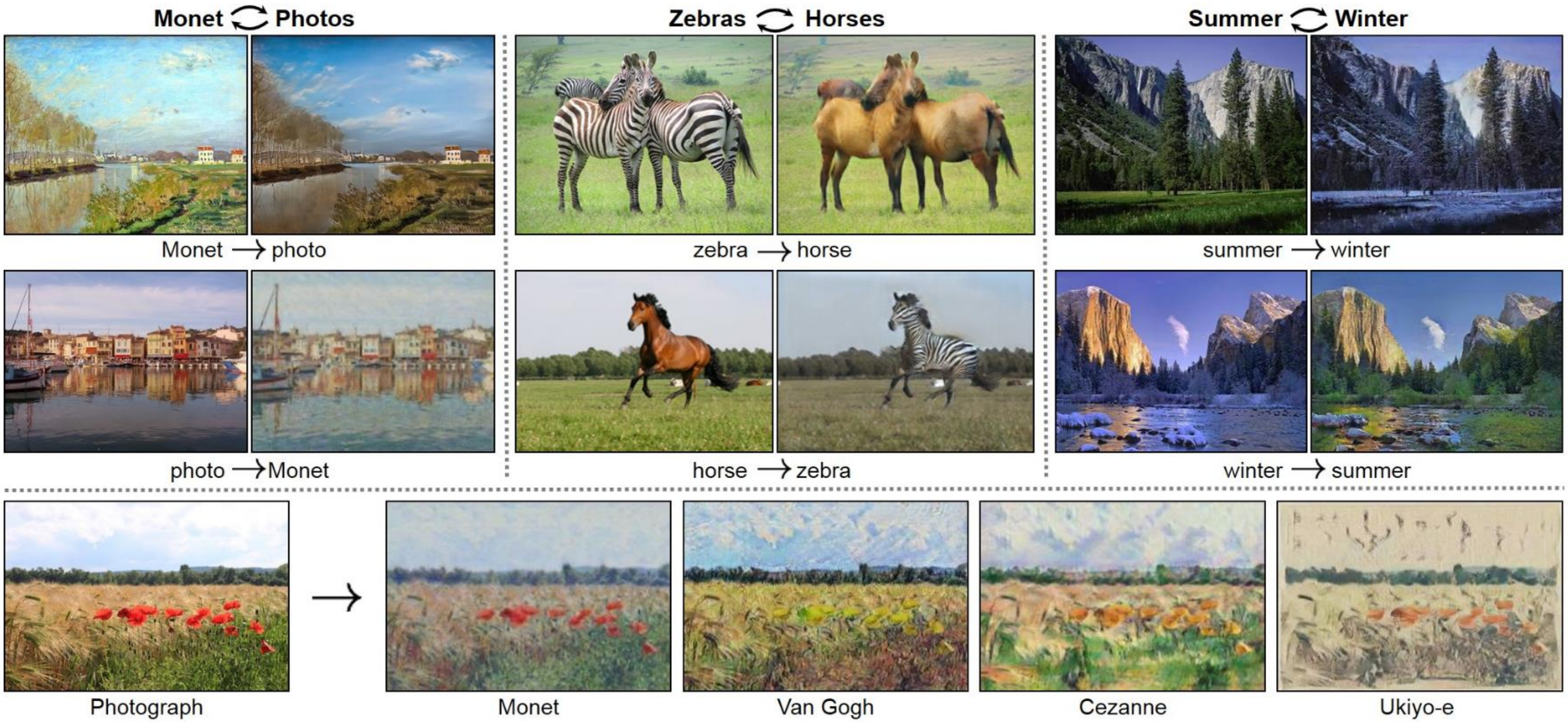
S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, H. Lee, [Generative adversarial text to image synthesis](), ICML 2016

Application: face generation



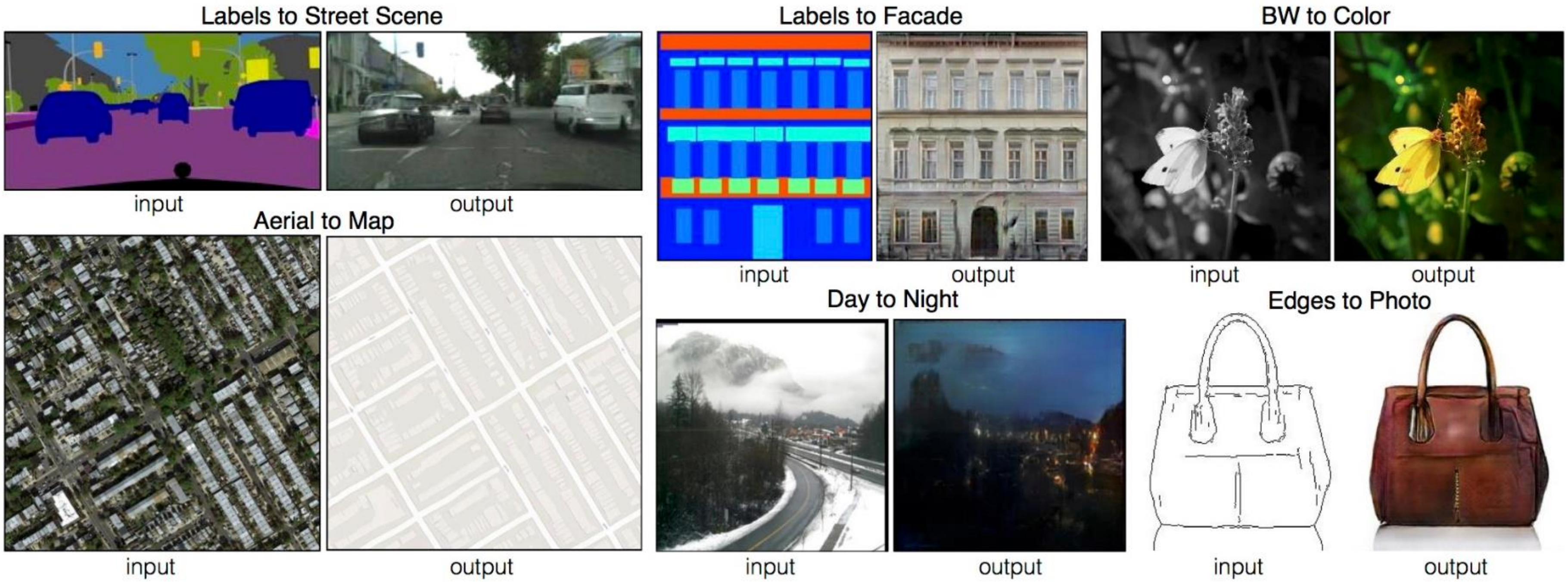
Source: https://www.youtube.com/watch?v=G06dEcZ-QTg

Application: style transfer



Source: https://github.com/junyanz/CycleGAN

Application: image translation



Source: https://github.com/phillipi/pix2pix

Conclusion

- GANs provide an attractive way to create a generative model without using an explicit encode (as in variational auto-encoders) but using supervised learning
- However the training is far from trivial and in general finding Nash equilibria in high-dimensional non-convex games is still an important open research problem