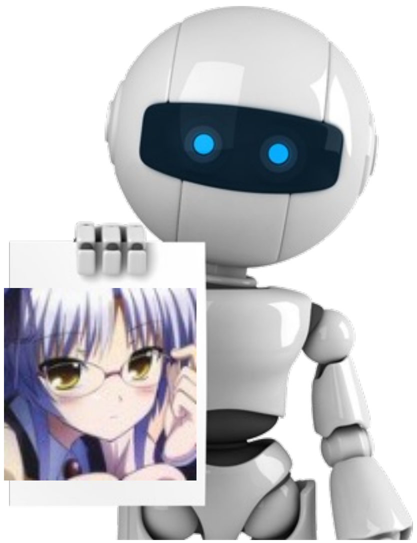
Generative models

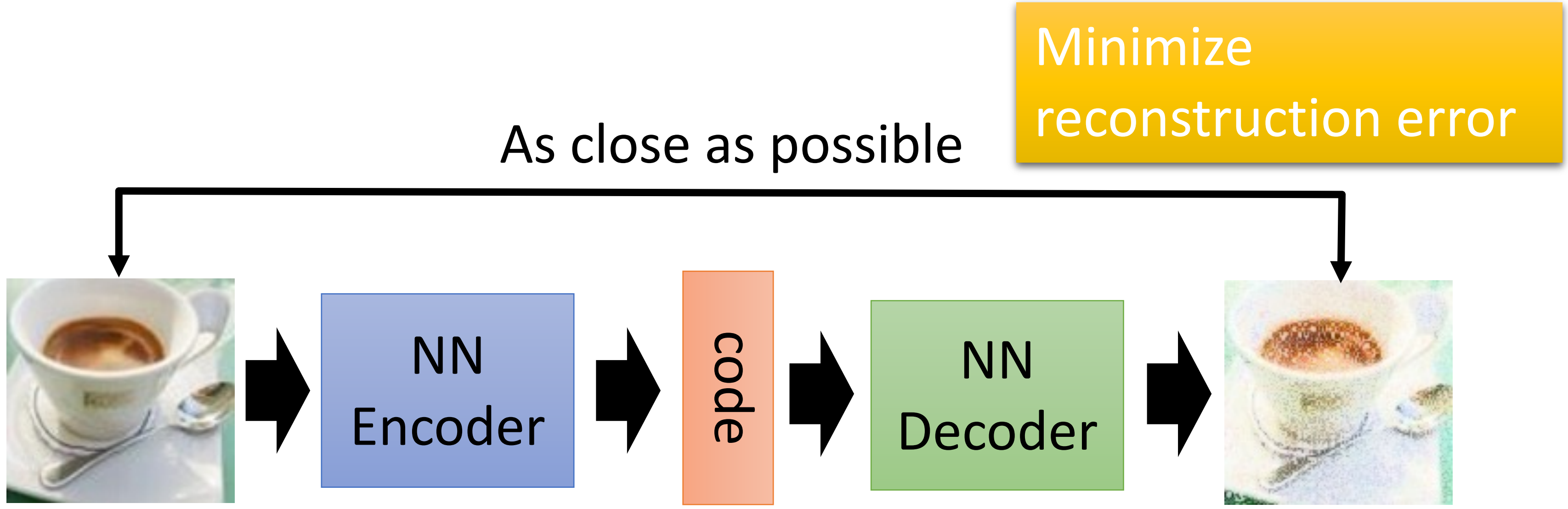
Outline

1. **Preview: Auto-Encoders, VAE**
2. Generative models with GAN
3. GAN architectures

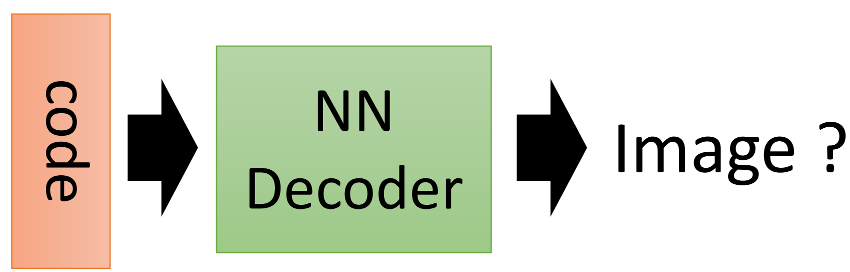


Drawing? => learning from examples

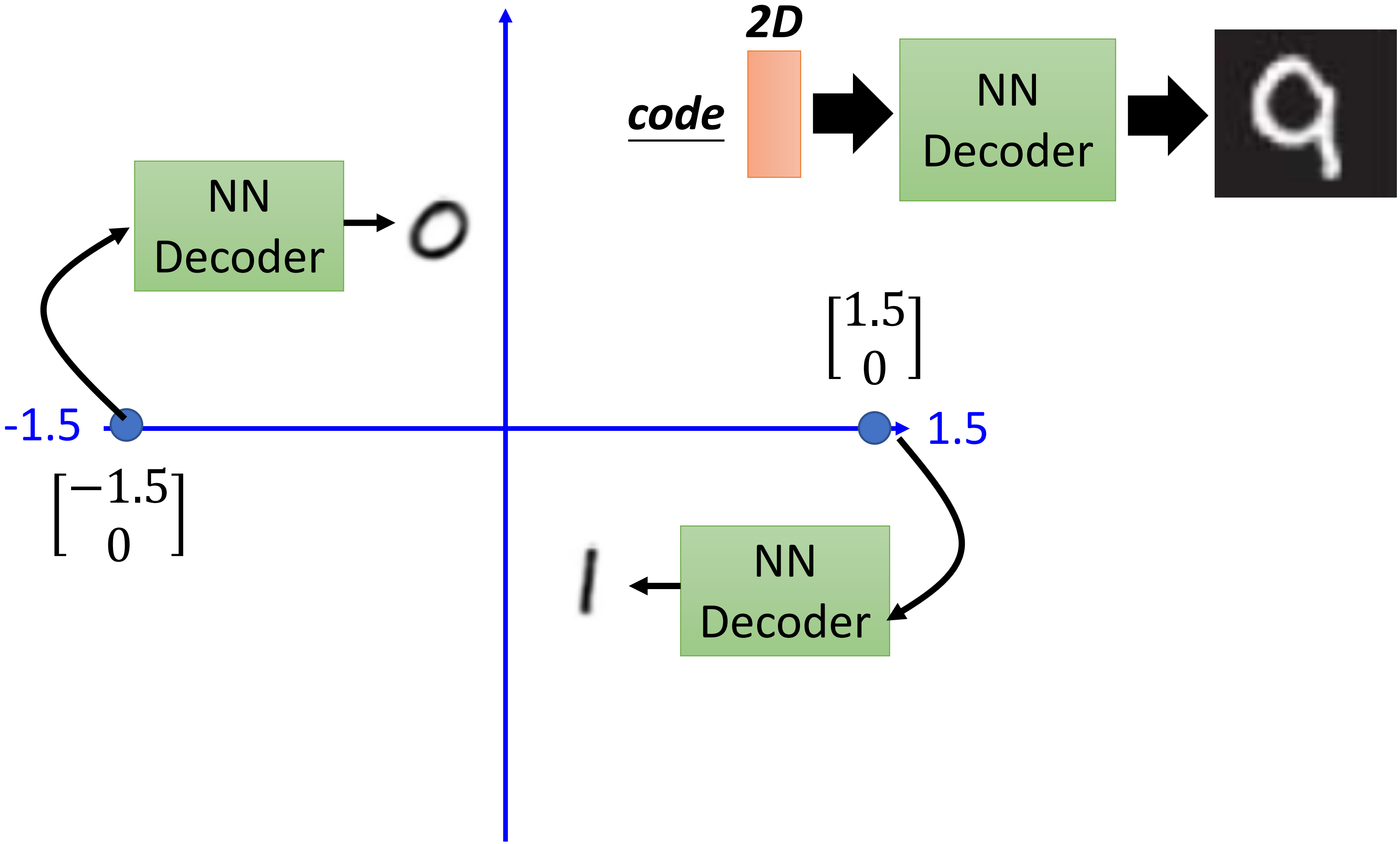
Review: Auto-encoder



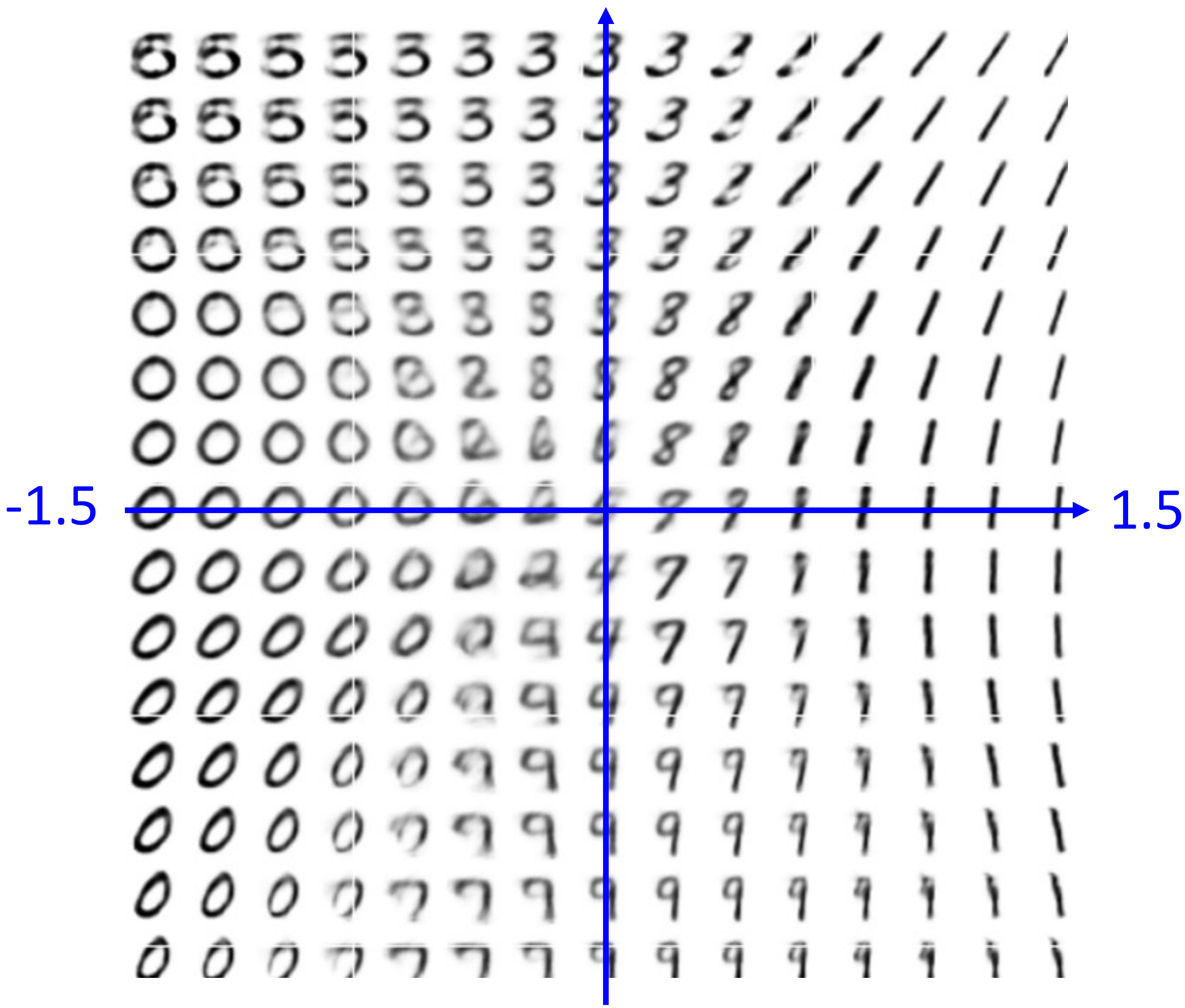
Randomly generate
a vector as code



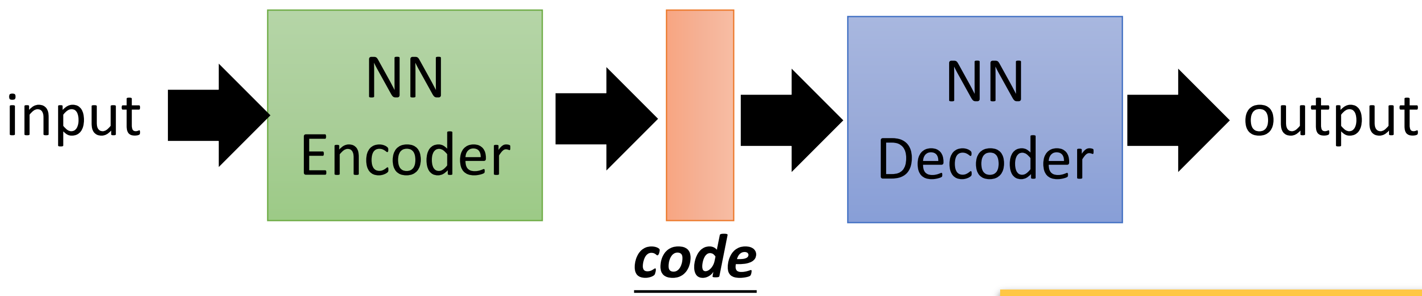
Review: Auto-encoder



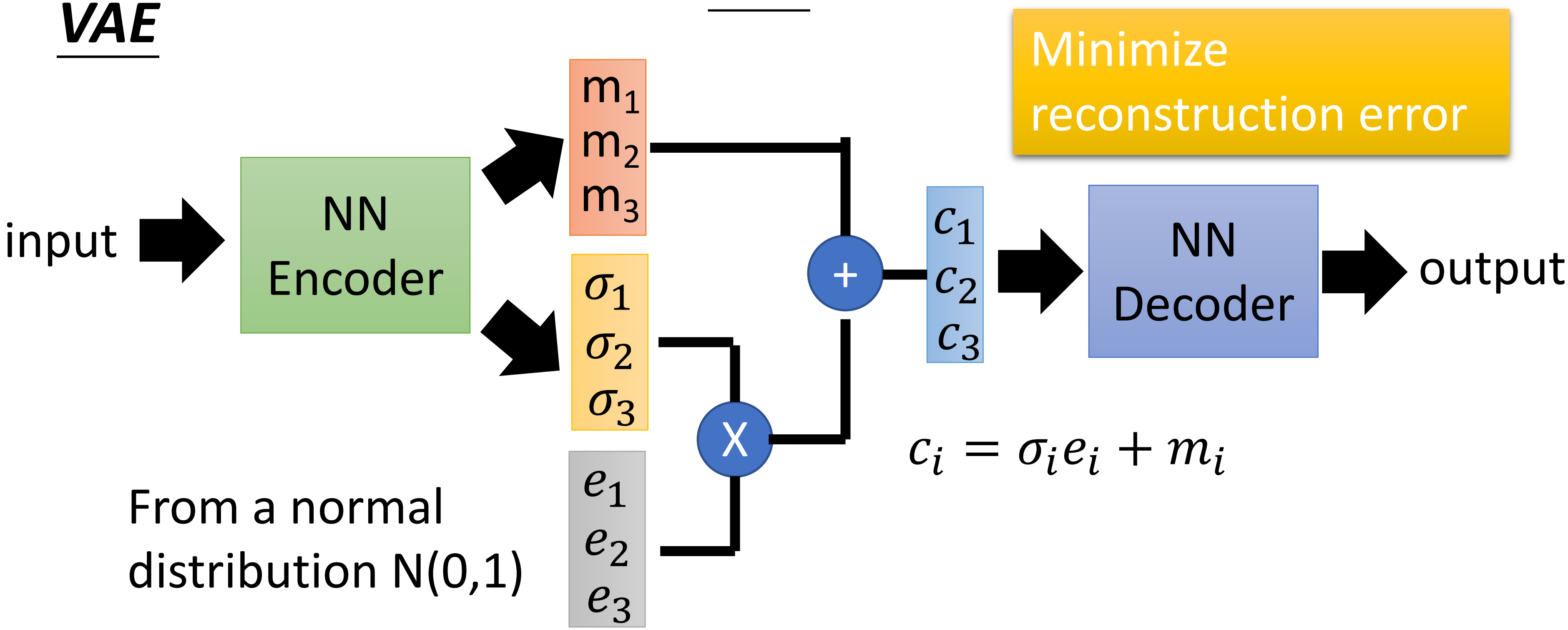
Review: Auto-encoder



Auto-encoder



VAE

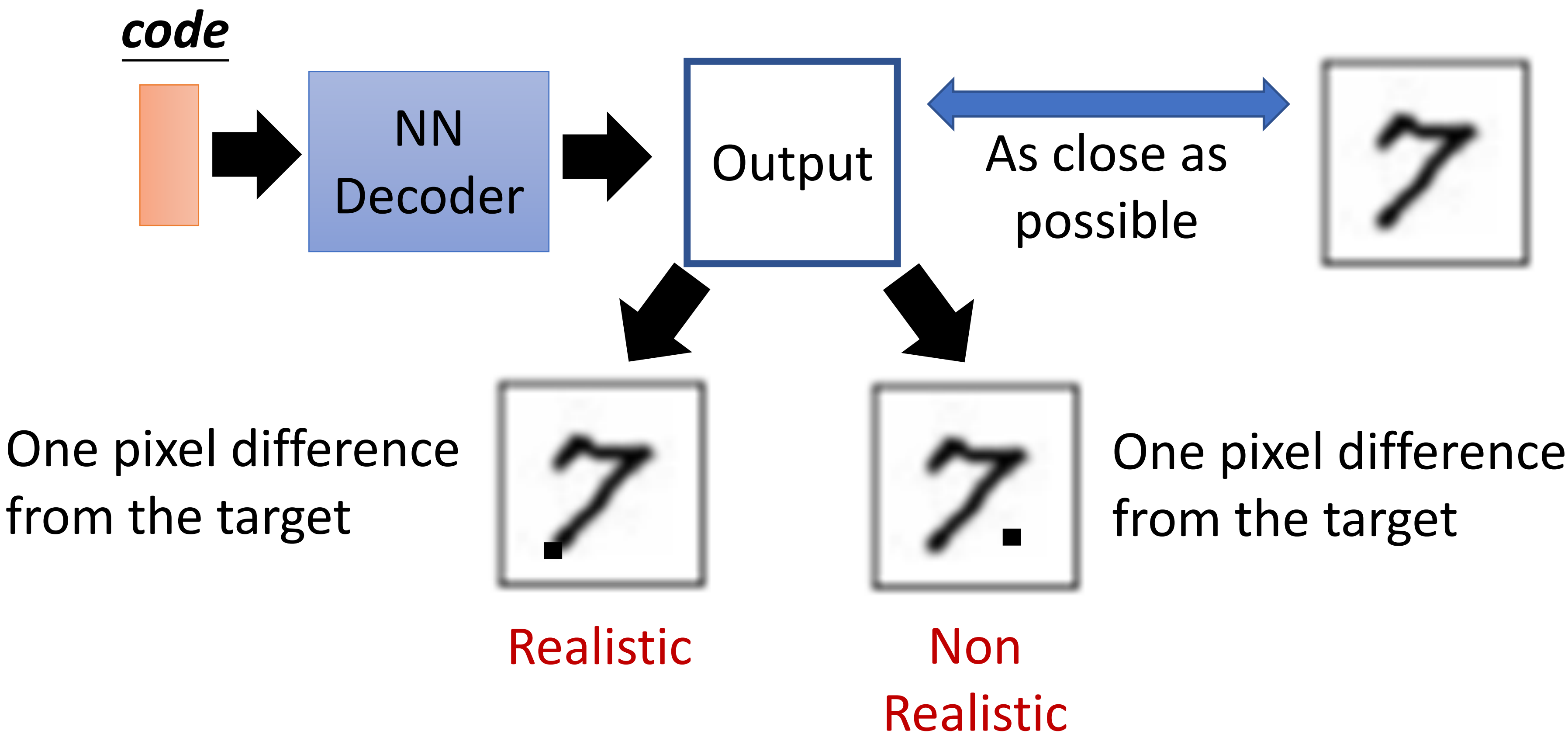


$$c_i = \sigma_i e_i + m_i$$

Auto-Encoding Variational Bayes, https://arxiv.org/abs/1312.6114

Problems of AE/VAE

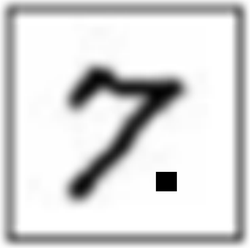
- It does not really try to simulate real images



Problems of AE/VAE

GAN to tackle this pb:

Realistic



Non Realistic

GAN: generative adversarial networks

Game scenario:

Player1, Generator, produces samples
Player2, – Its adversary **Discriminator**, attempts to distinguish real samples from fake generated ones (produced by Player1) !

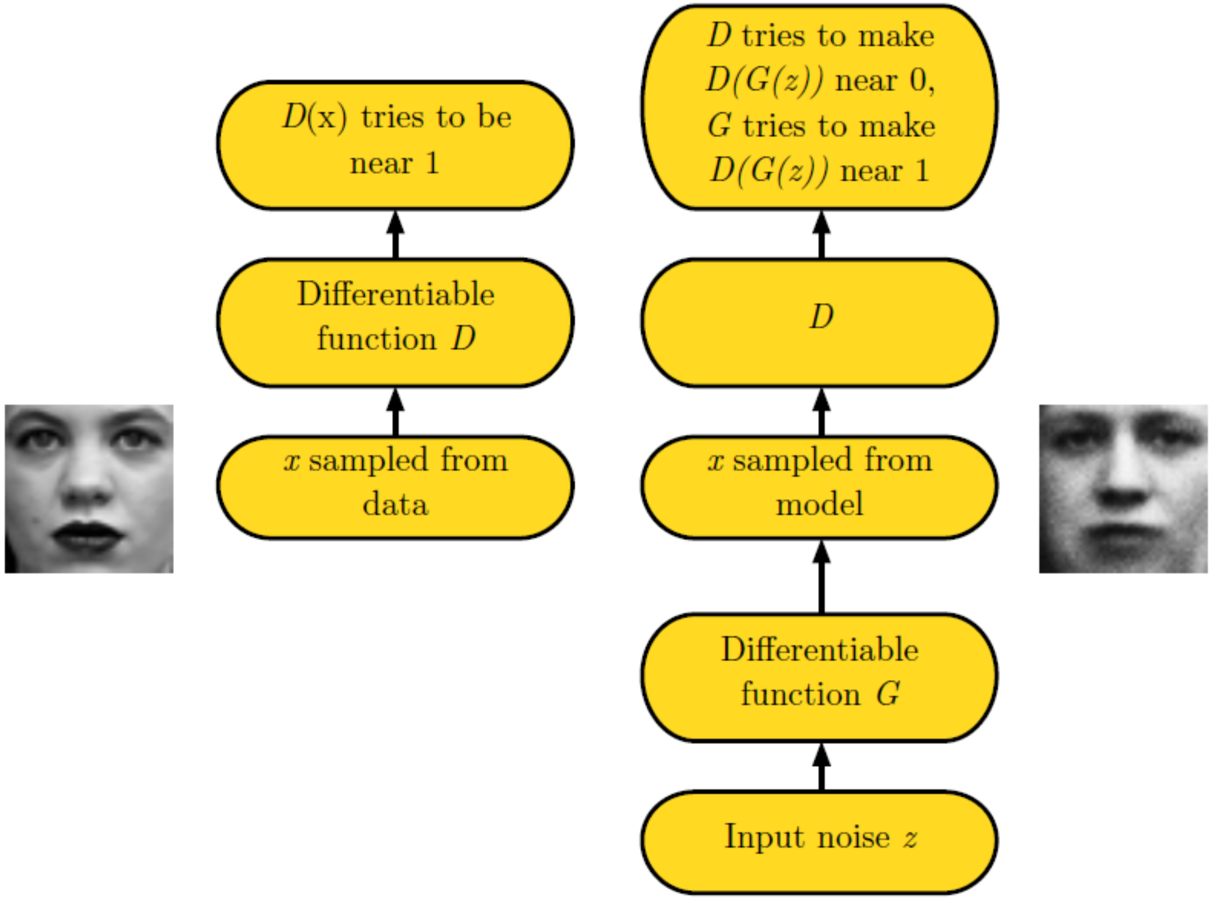
Player1 aims at producing Realistic images to fool the Player2

Generative models

Outline

1. Preview: Auto-Encoders, VAE
2. **Generative models with GAN**
    - GAN Algorithm

Adversarial Nets Framework

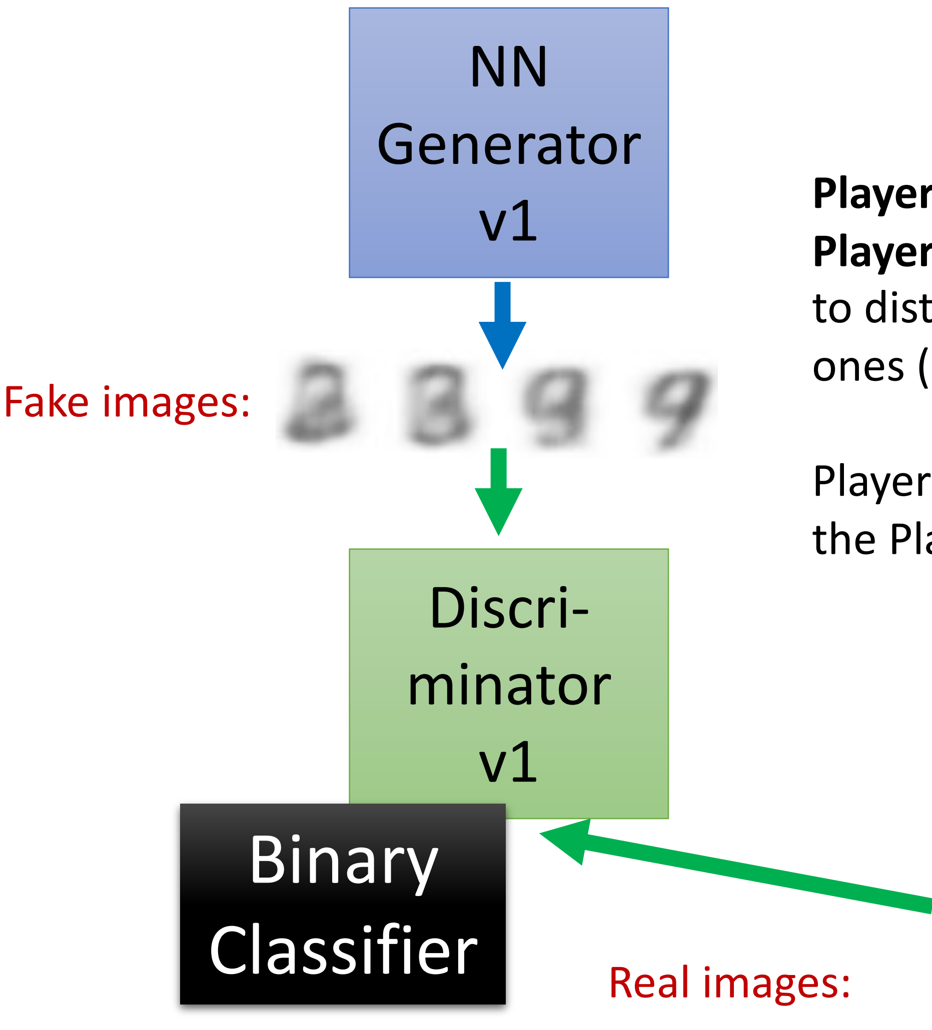


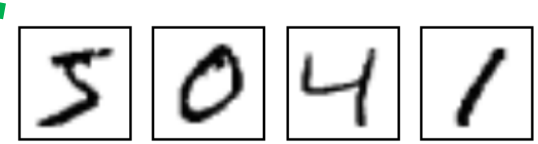
(Goodfellow 2016)

$$V = \mathbb{E}_{x \sim P_{data}}[logD(x)] + \mathbb{E}_{x \sim P_G}\left[log\left(1 - D(x)\right)\right]$$

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

GAN Learning – D and G updates



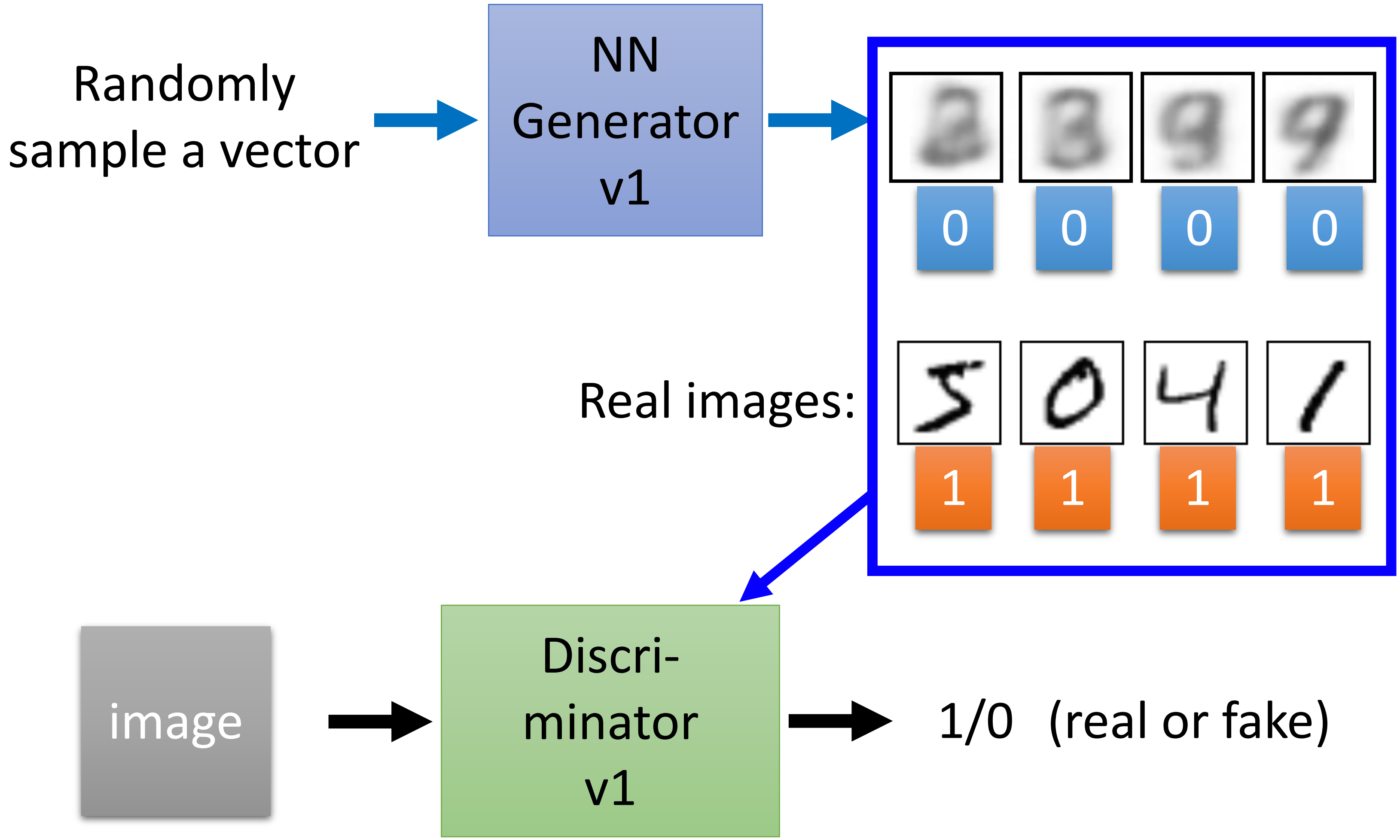


Game scenario:

Player1, Generator G, produces samples
Player2, – Its adversary **Discriminator D**, attempts to distinguish real samples from fake generated ones (produced by Player1) !

Player1 aims at producing Realistic images to fool the Player2

GAN - Discriminator



Discriminator Optimization on a batch of images:

Using gradient descent to update the parameters in the discriminator, with a fixed generator

GAN - Generator

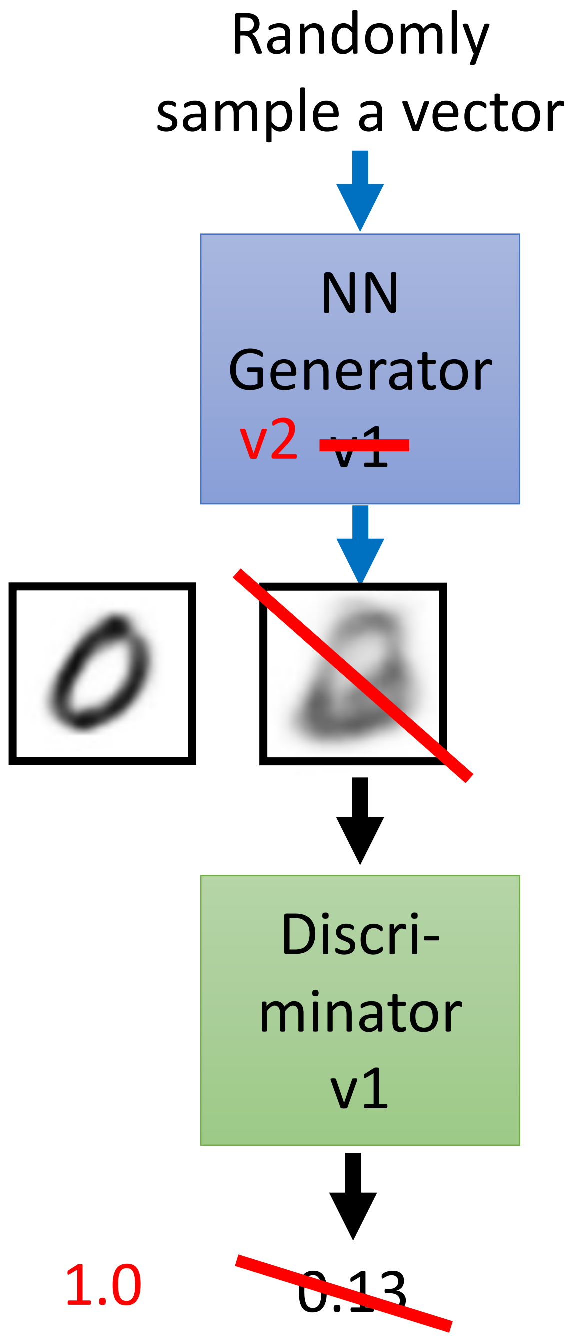
Updating the parameters of generator

The output be classified as "real" (as close to 1 as possible)

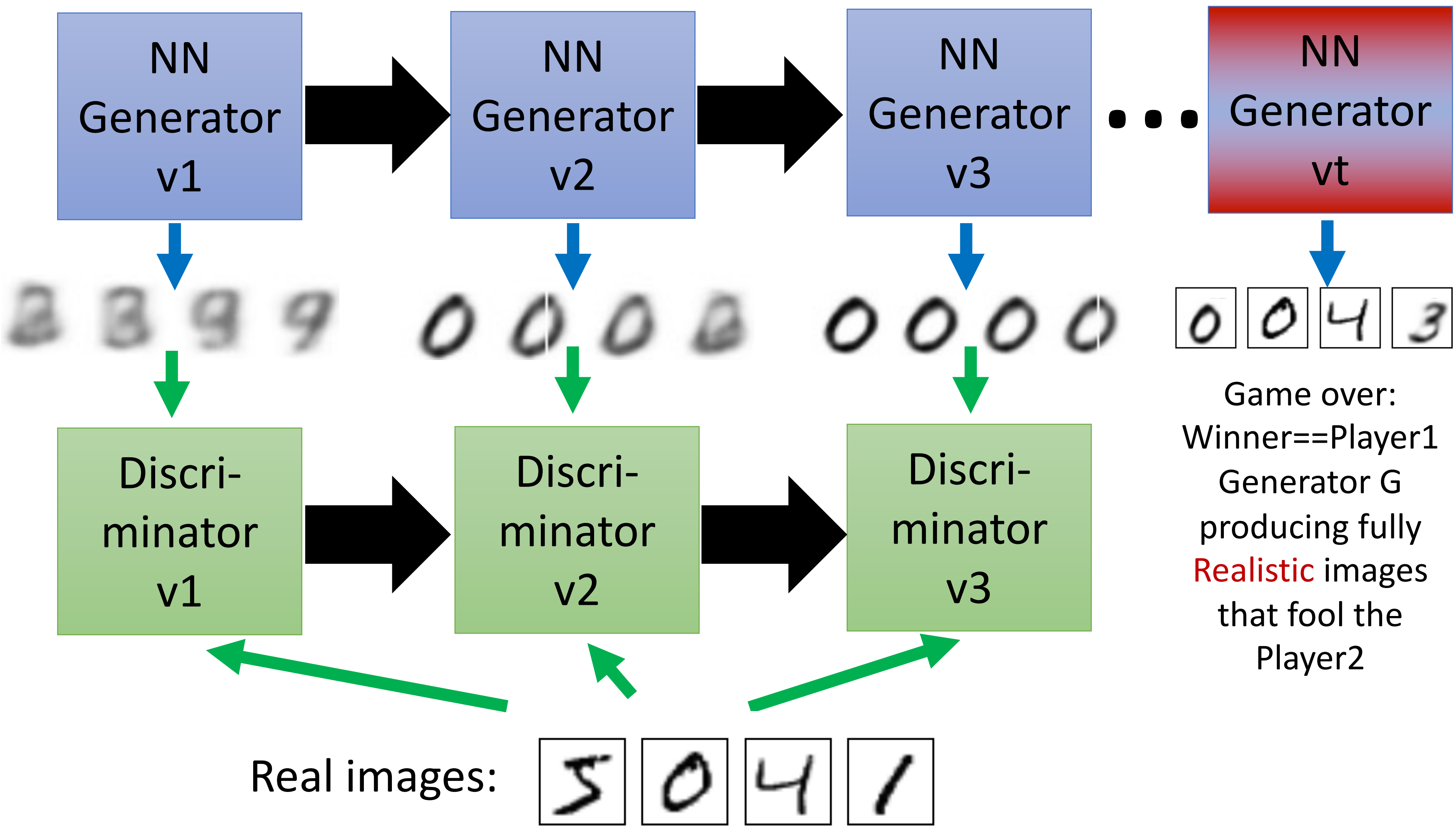
Generator + Discriminator = a network

Optimization:

Using gradient descent to update the parameters in the generator, but fixing the discriminator



GAN Learning – D and G updates



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:

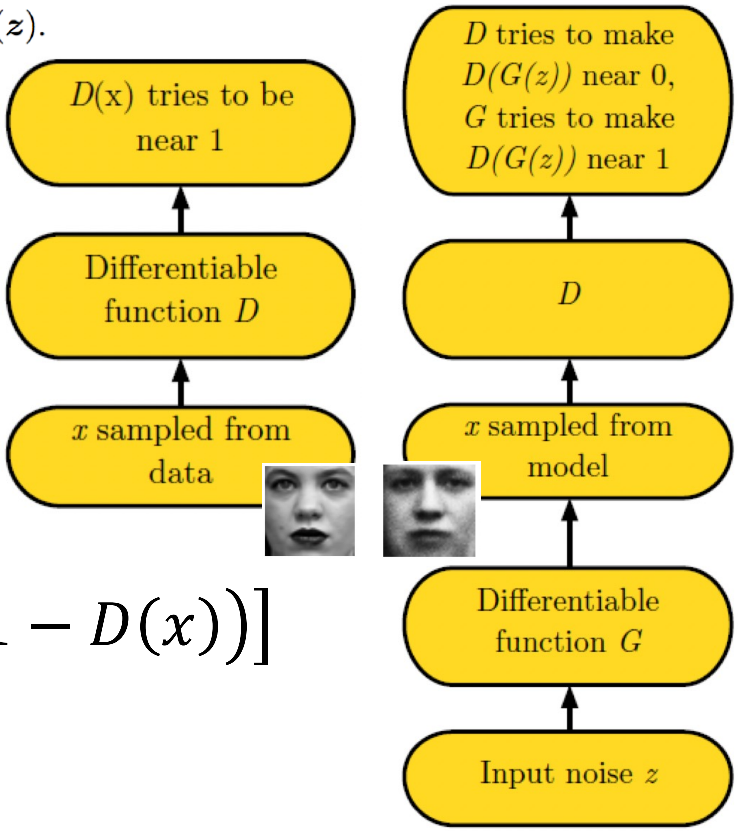
$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  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

GAN algorithm



$$V = \mathbb{E}_{x \sim P_{data}}[logD(x)] + \mathbb{E}_{x \sim P_G}[log(1 - D(x))]$$

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

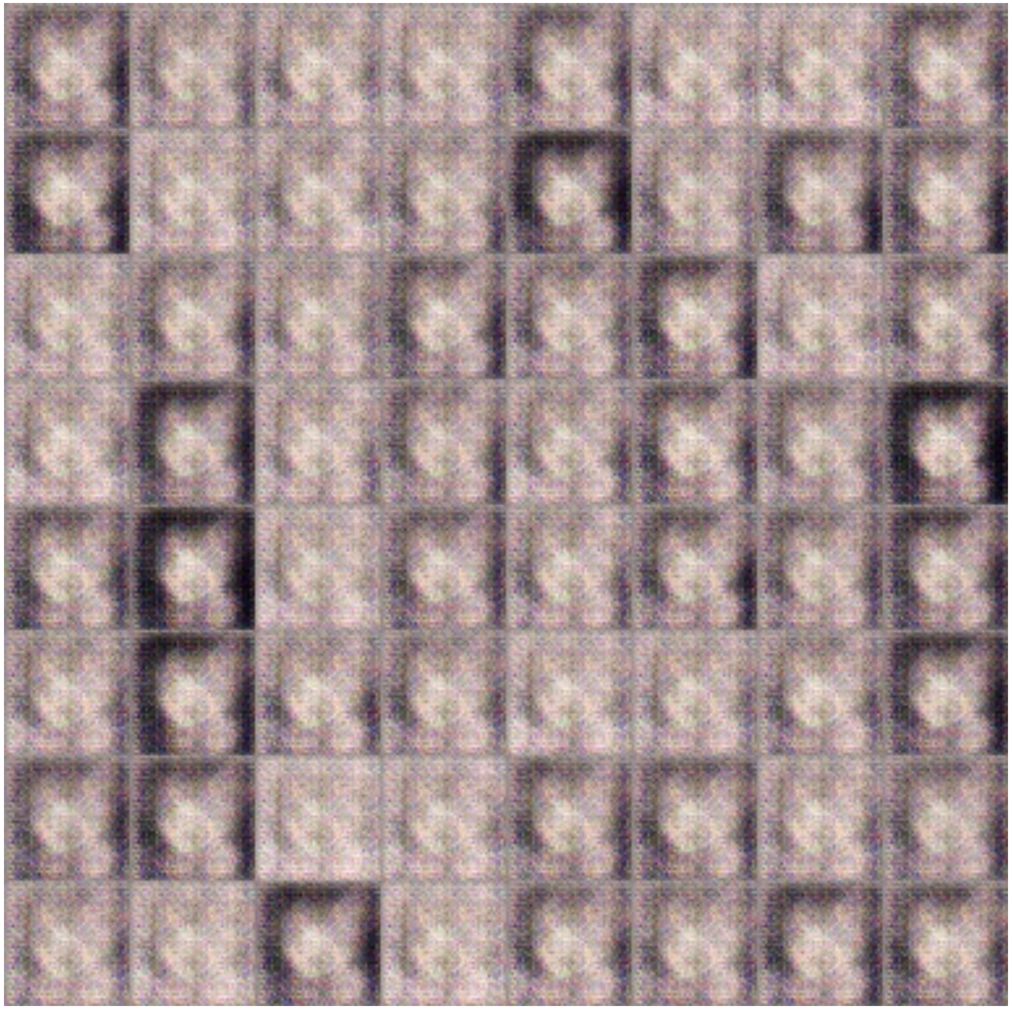
One example GAN

Source of images: https://zhuanlan.zhihu.com/p/24767059

DCGAN: https://github.com/carpedm20/DCGAN-tensorflow

GAN

100 rounds



GAN

20,000 rounds

GAN

50,000 rounds

Generative models

Outline

1. Preview: Auto-Encoders, VAE
2. Generative models with GAN
3. **GAN architectures**

Drawing? => learning from examples

Recall Algo GAN

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:

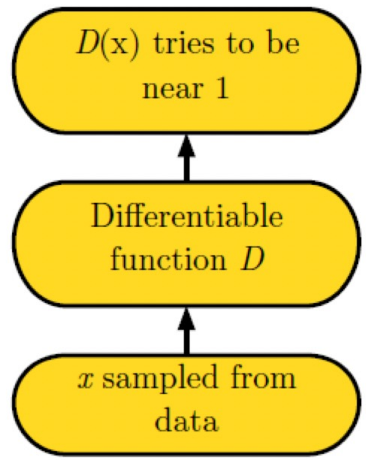
$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

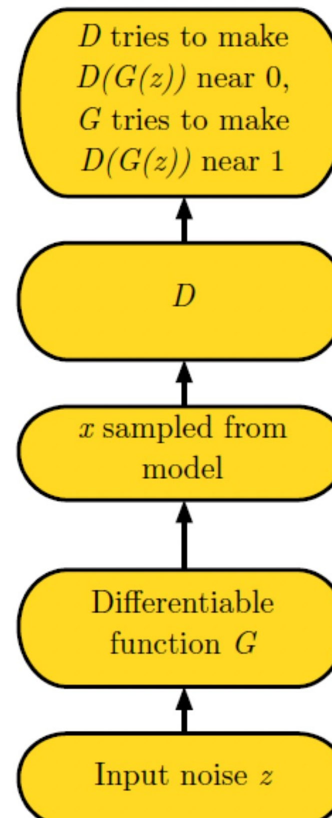
end for

- 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





Functions G and D are NN

Question:

Which architectures for G and D?

Generative models

Outline

1. Preview: Auto-Encoders, VAE
2. Generative models with GAN
3. **GAN architectures**
    1. **Basics**

Basic Archi for G and D and expe

Models

G and D fully connected nets
or convolutional for D, (de)convolutional for G (as seen for segmentation nets)
ReLU and/or sigmoids, dropout

Datasets

MNIST, Toronto Face Database, CIFAR-10

GAN - Evaluation

- Approximate p_g by fitting a Gaussian Parzen window on the generated images.
- Cross-validate σ to maximize likelihood of validation set
- Compute the likelihood of the test set

Evaluation not trivial, can be done using generated images as inputs for deep nets => inception scores

GAN - Qualitative results 1/2

Figure: Right col nearest from dataset. a) MNIST, b) TFD, c) CIFAR-10 (fully connected), d) CIFAR-10 (convolutional D, deconvolutional G)

GAN - Qualitative results 2/2

Figure: Linear interpolation between 2 points in z space

- **Advantages:**
  - Computational advantages (no complex likelihood inference)
  - Can represent sharper distributions
- **Disadvantages:**
  - G and D must be well synchronized for the algorithm to converge correctly

GAN architectures

- How to improve result quality?
  - Spatial resolution

    $\Rightarrow$Cascade of GAN
  - Object quality

    => Progressive growing of spatial resolution in G

Generative models

Outline

1. Preview: Auto-Encoders, VAE
2. Generative models with GAN
3. GAN architectures
    1. Basics
    2. **LaPGAN**

Laplacian Pyramid GANs (LAPGANs)

- GANs do not work well for complex / high level / natural images.
- Idea: decompose the generation in successive tasks using Laplacian Pyramid (of GANs)

Let $d(I)$ and $u(I)$ be down-sampling and up-sampling operations.
Gaussian pyramid:

$$\mathcal{G}(I) = [I_0, I_1, ..., I_K], I_k = d^{(k)}(I)$$

Laplacian pyramid:

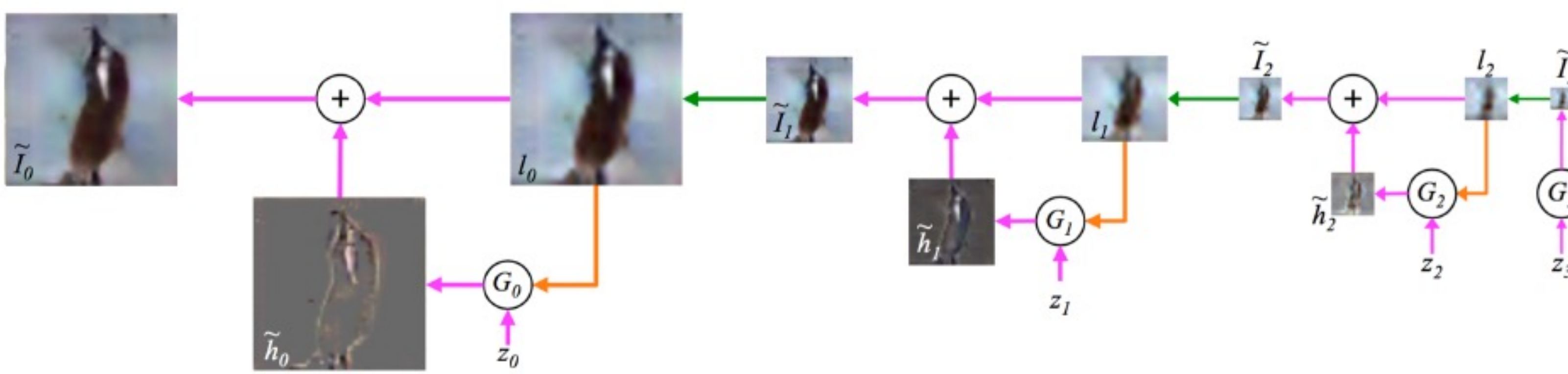
$$h_k = \mathcal{L}_k(I) = I_k - u(I_{k+1})$$

Reconstruction:

$$I_k = u(I_{k+1}) + h_k$$

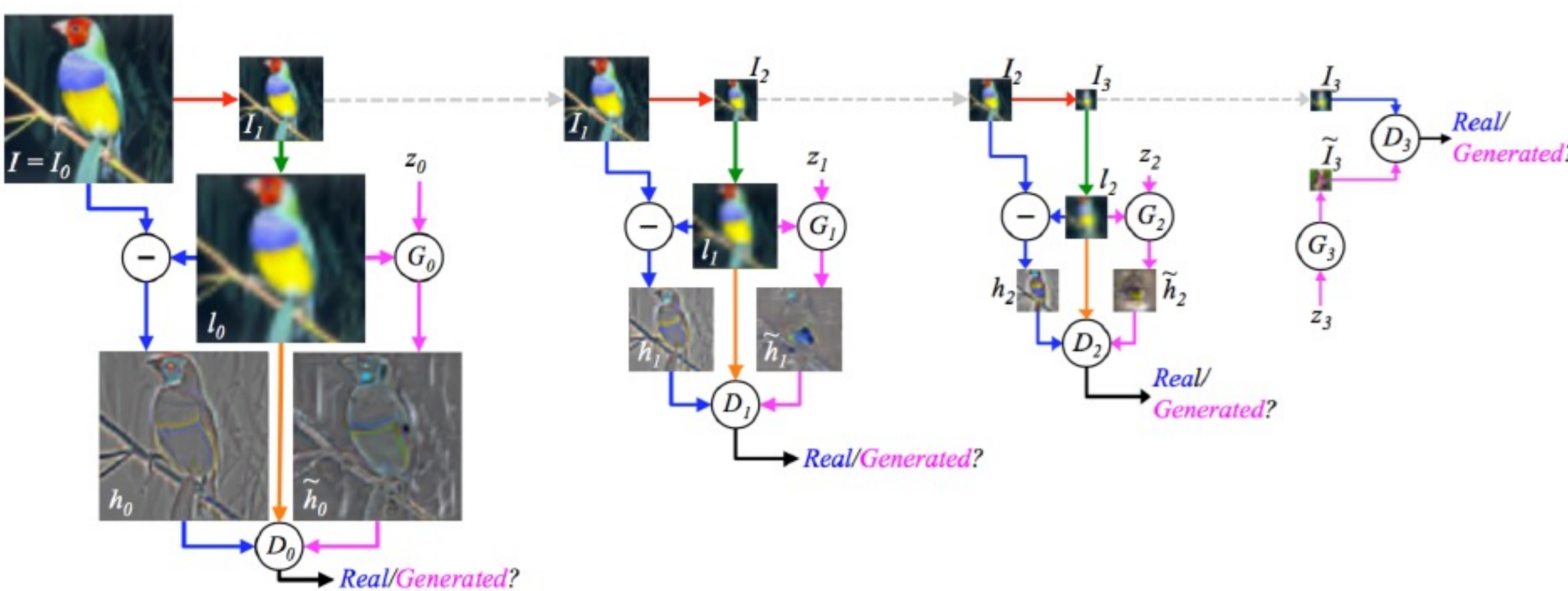
LAPGAN model - sampling

- Set of generative convnets: $G_0, ..., G_K$
- Generated details: $\tilde{h}_k = G_k(z_k, u(\tilde{I}_{k+1}))$
- Reconstructed image: $\tilde{I}_k = u(\tilde{I}_{k+1}) + \tilde{h}_k$ ($\tilde{I}_{K+1} = 0$)



LAPGAN model - training

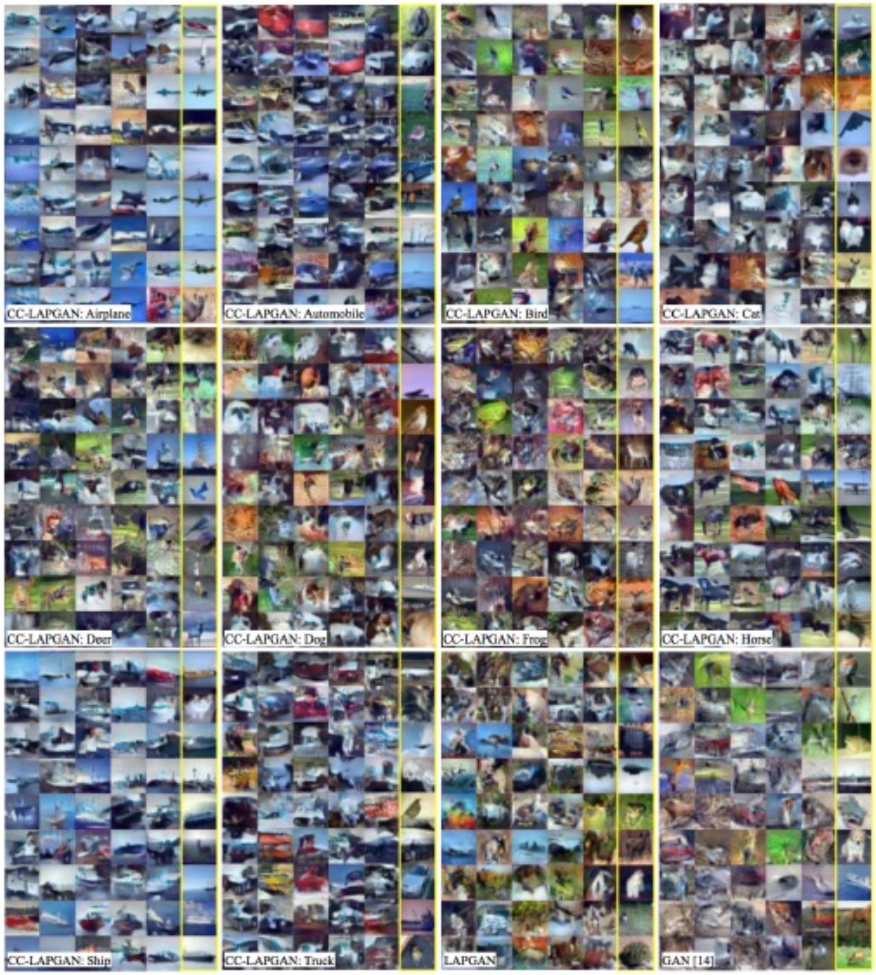
- Low-pass version of I_0: $l_0 = u(d(I_0))$
- Either:
  - High-pass version of I_0: $h_0 = I_0 - l_0$
  - Generate $\tilde{h}_0 = G_0(z_0, l_0)$
- Forward $D_0(l_0 + h_0 \text{ or } \tilde{h}_0)$
- Backward D_0 and G_0
- G_K and D_K are trained as a simple GAN



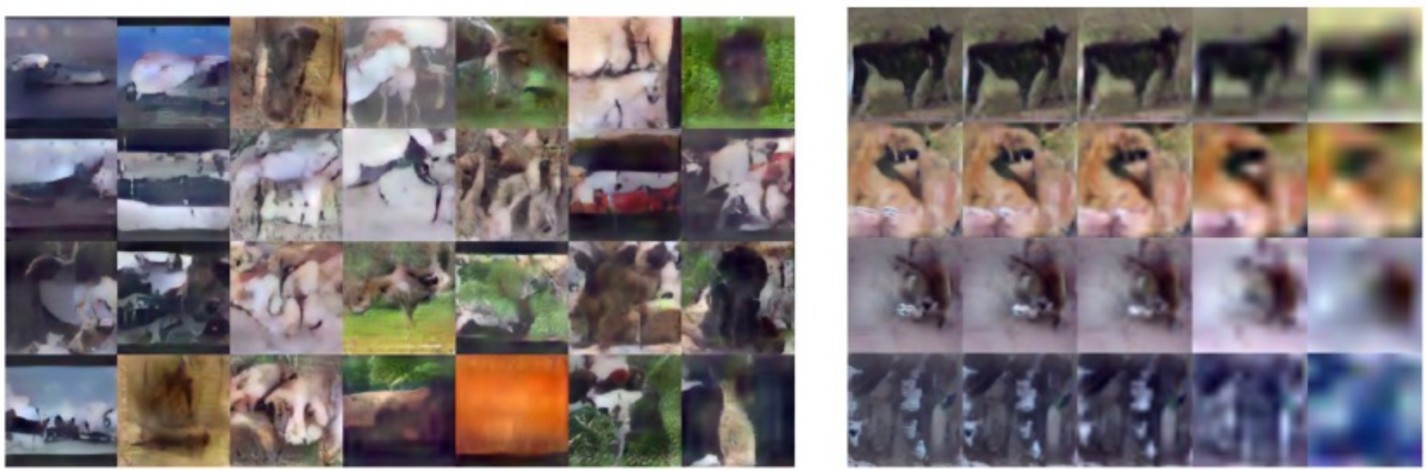
LAPGAN model - Experiments

- **Datasets:** CIFAR-10, STL
- **Initial scale:**
  - G_K and D_K have 2 hidden layers and ReLU
  - $z_K \sim U_{[-1,1]^{100}}$
  - Trained as GAN
- **Subsequent scales:**
  - G_k and D_k convnets with 3 and 2 layers
  - $z_k \sim U_{[-1,1]^{|I_k|}}$ ("color" layer)
  - Trained as CGAN

LAPGAN model - Results - CIFAR



LAPGAN model - Results - STL



(a) (b)

Figure 4: STL samples: **(a)** Random 96x96 samples from our LAPGAN model. **(b)** Coarse-to-fine generation chain.

LAPGAN model - Results - LSUN

LAPGAN

- Good idea (cascade) but Generator structure too weak

=> Other approach: progressive growing of spatial resolution

Generative models

Outline

1. Preview: Auto-Encoders, VAE
2. Generative models with GAN
3. GAN architectures
    1. Basics
    2. LaPGAN
    3. **DCGAN**

Progressive growing of spatial resolution in G

Deep Convolutional GANs (DCGANs)

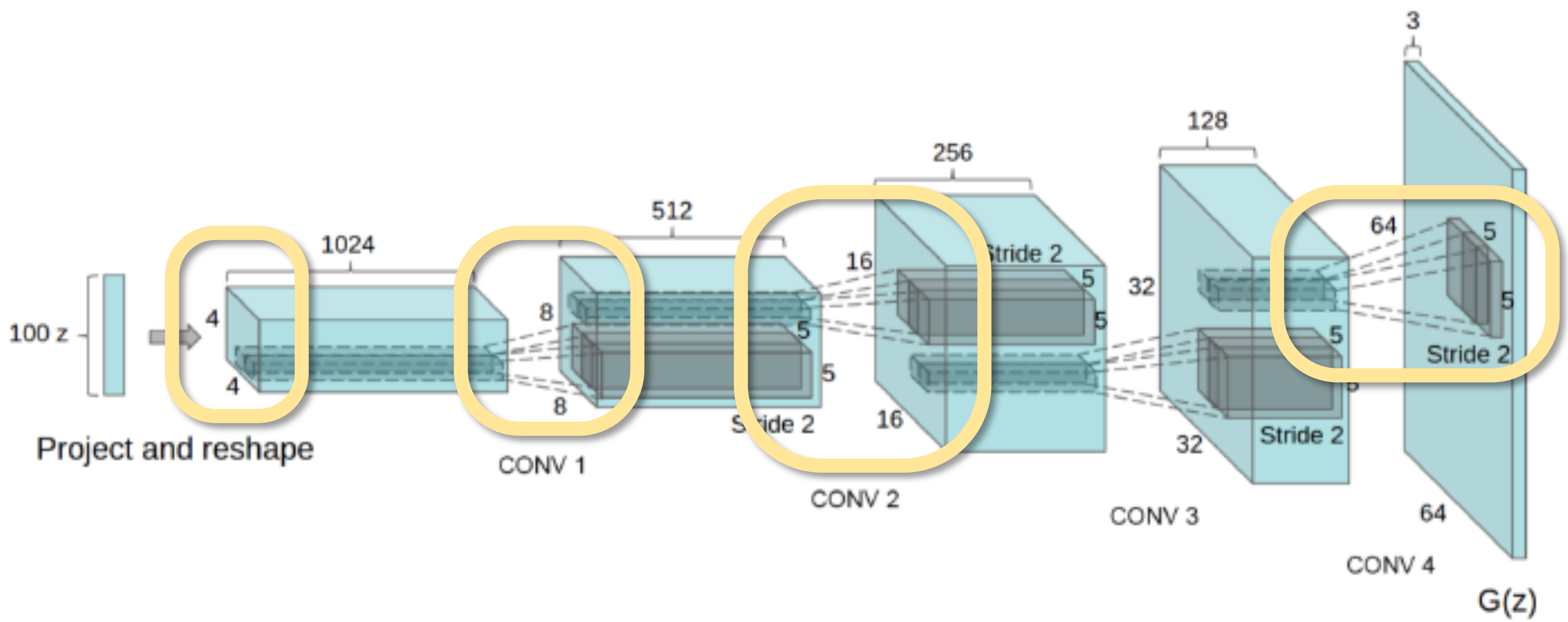
GANs are hard to scale => Identify a family of architectures that gives stable training

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator) =transposed convolutions =deconvolutions
- Use batchnorm in both the generator and the discriminator
- Remove fully connected hidden layers for deeper architectures
- Use ReLU activation in generator for all layers except for the output, which uses Tanh
- Use LeakyReLU activation in the discriminator for all layers

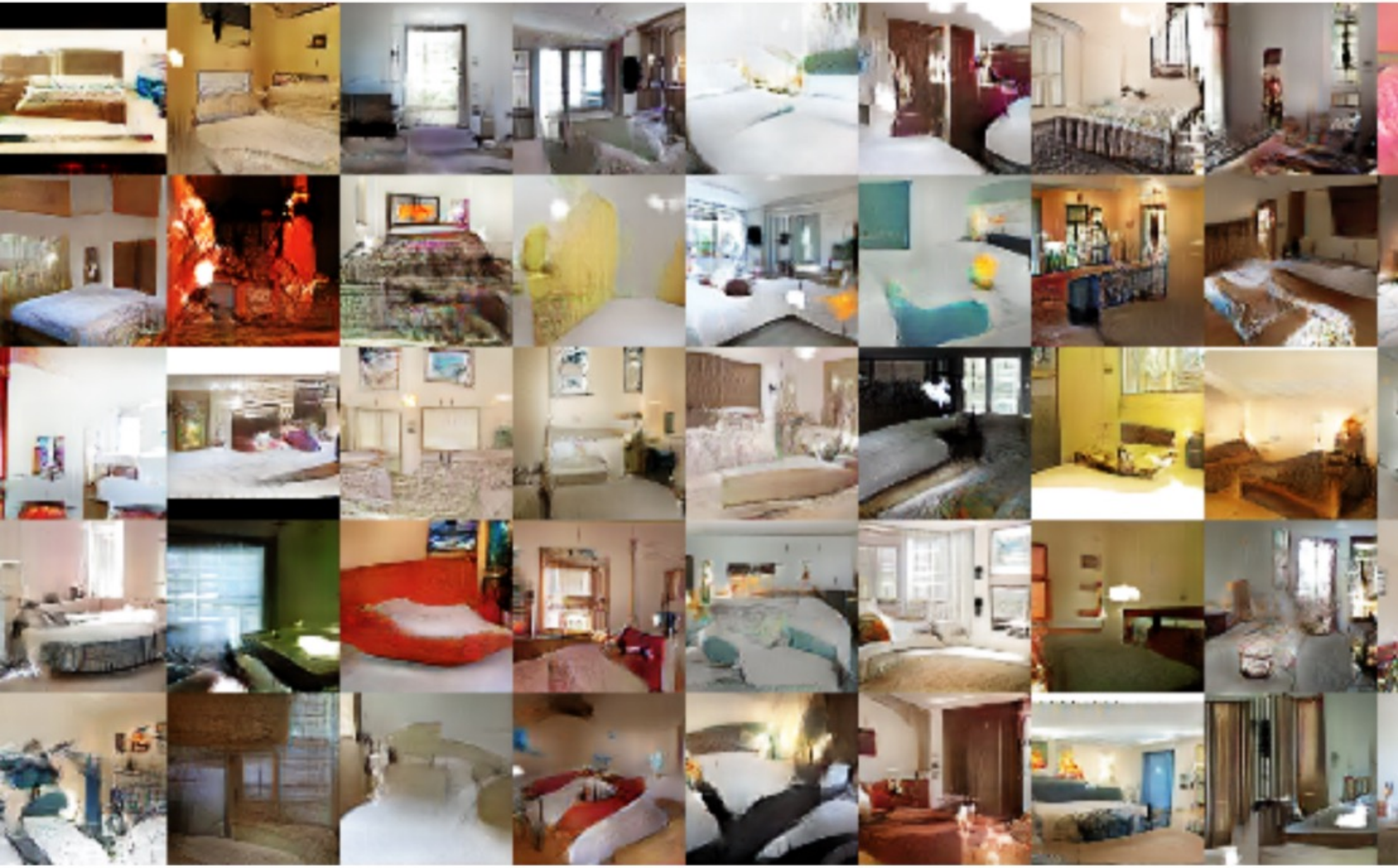
Progressive growing of spatial resolution in G: DCGAN

Upsampling step by step

Combine with convolutional layers



DCGAN - Results - generated bedrooms



DCGAN results - Faces

Generative models

Outline

1. Preview: Auto-Encoders, VAE
2. Generative models with GAN
3. GAN architectures
    1. Basics
    2. LaPGAN
    3. DCGAN
    4. **ProGAN**

Progressive growing of GANs

[Progressive Growing of GANs for Improved Quality, Stability and Variation, Tero Karras et al. (NVIDIA); ICLR 2018]

1. First, start with training 4x4 output images.
2. When this training has converged, add a new block to generate 8x8 output images.
3. Etc.

The transition to adding a new block is gradual, we first start with more weight on the (upsampled) output of the previous block, and then add more and more weight to the output of the current block.

All weights remain trainable during the whole process.

Discriminator = mirror image of generator

Progressive growing of GANs

[Progressive Growing of GANs for Improved Quality, Stability and Variation, Tero Karras et al. (NVIDIA); ICLR 2018]

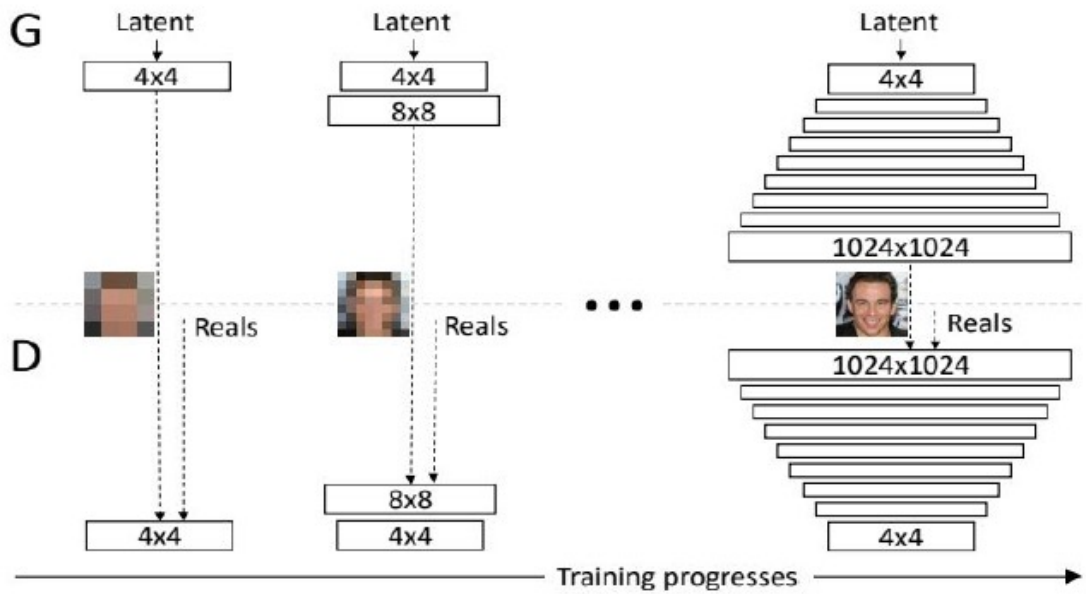


Figure 1: Our training starts with both the generator (G) and discriminator (D) having a low spatial resolution of 4×4 pixels. As the training advances, we incrementally add layers to G and D, thus increasing the spatial resolution of the generated images. All existing layers remain trainable throughout the process. Here $\boxed{N \times N}$ refers to convolutional layers operating on $N \times N$ spatial resolution. This allows stable synthesis in high resolutions and also speeds up training considerably. One the right we show six example images generated using progressive growing at 1024×1024.

Progressive growing of GANs

[Progressive Growing of GANs for Improved Quality, Stability and Variation, Tero Karras et al. (NVIDIA); ICLR 2018]

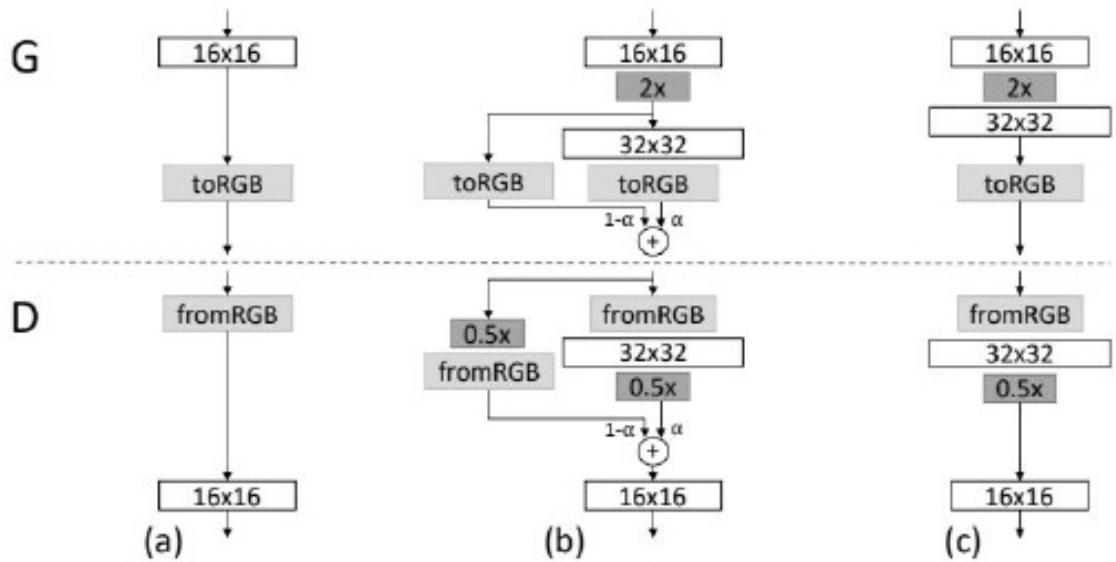


Figure 2: When doubling the resolution of the generator (G) and discriminator (D) we fade in the new layers smoothly. This example illustrates the transition from 16×16 images (a) to 32×32 images (c). During the transition (b) we treat the layers that operate on the higher resolution like a residual block, whose weight α increases linearly from 0 to 1. Here $\boxed{2\times}$ and $\boxed{0.5\times}$ refer to doubling and halving the image resolution using nearest neighbor filtering and average pooling, respectively. The $\boxed{\text{toRGB}}$ represents a layer that projects feature vectors to RGB colors and $\boxed{\text{fromRGB}}$ does the reverse; both use 1×1 convolutions. When training the discriminator, we feed in real images that are downscaled to match the current resolution of the network. During a resolution transition, we interpolate between two resolutions of the real images, similarly to how the generator output combines two resolutions.

Generative models

Outline

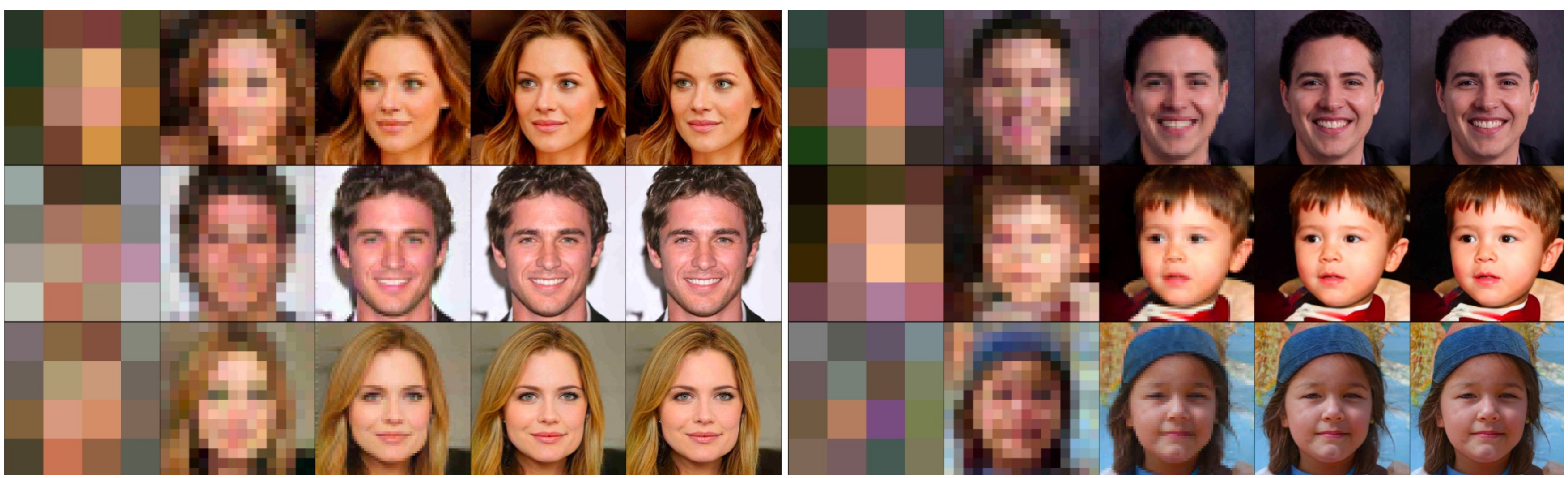
1. Preview: Auto-Encoders, VAE
2. Generative models with GAN
3. GAN architectures
    1. Basics
    2. LaPGAN
    3. DCGAN
    4. ProGAN
    5. **MSG-GAN**

MSG-GAN: Multi-Scale Gradients for Generative Adversarial Networks [CVPR 2020]

Main Idea:

- ProGAN both use progressive growing, but although this gives stability, it introduces many complicated training parameters associated with each new network.
- Training cannot be done "out of the box", have to adjust parameters for each new dataset.

→ Train all at once without complicated adding on layers



MSG-GAN: Multi-Scale Gradients for Generative Adversarial Networks [CVPR 2020]

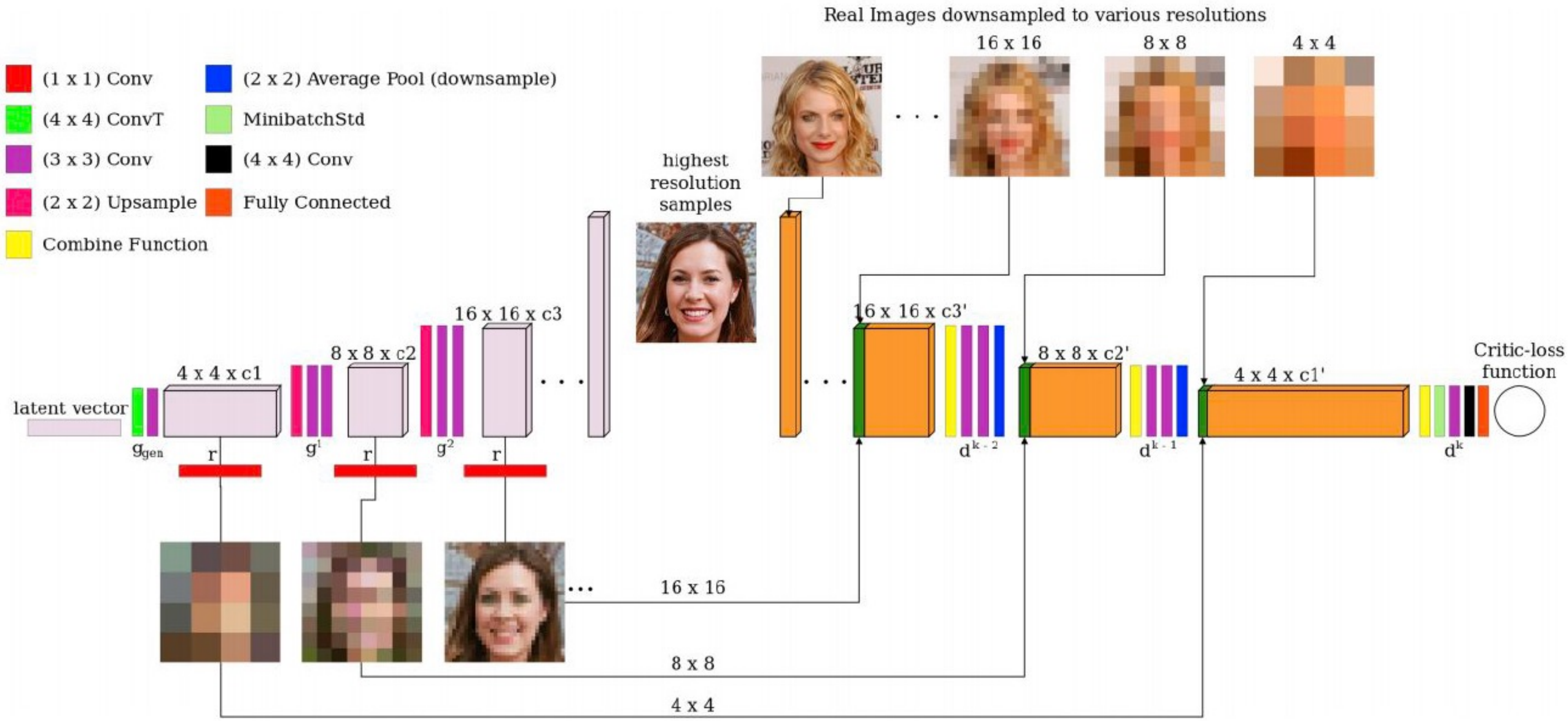


Figure 2: Architecture of MSG-GAN, shown here on the base model proposed in ProGANs [13]. Our architecture includes connections from the intermediate layers of the generator to the intermediate layers of the discriminator. Multi-scale images sent to the discriminator are concatenated with the corresponding activation volumes obtained from the main path of convolutional layers followed by a combine function (shown in yellow).

MSG-GAN: results – Random generated CelebA-HQ Faces at resolution 1024x1024

Generative models

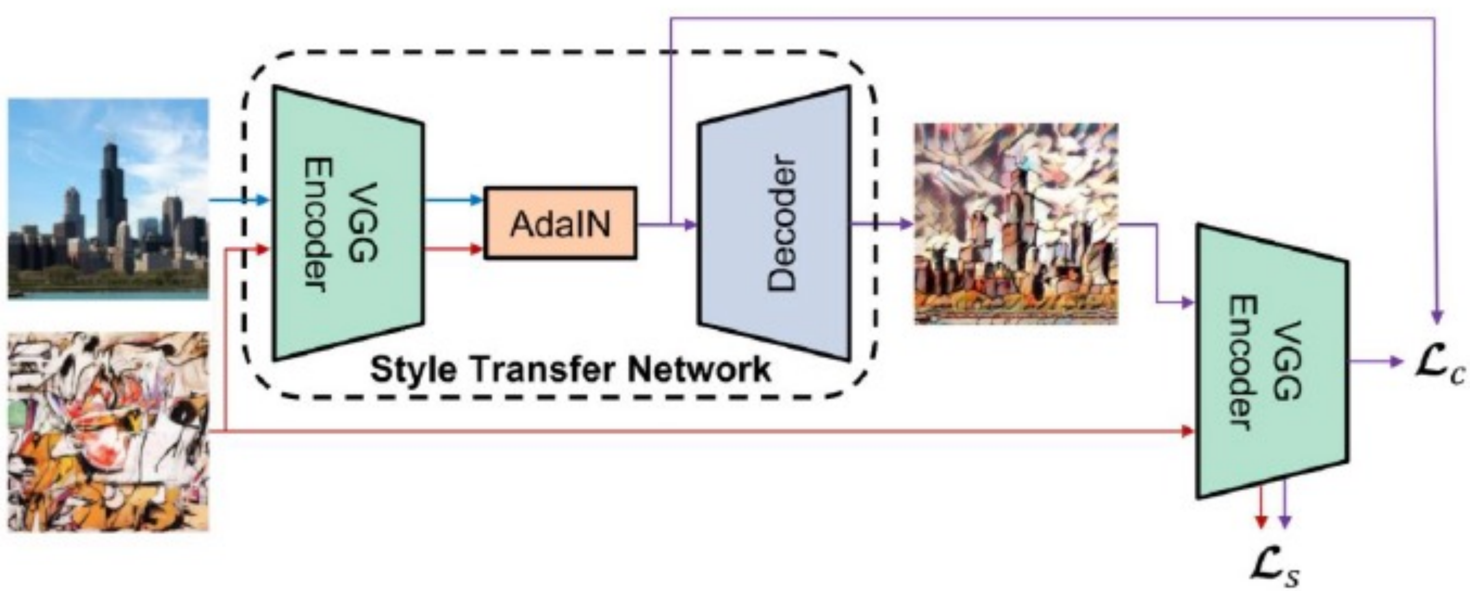
Outline

1. Preview: Auto-Encoders, VAE
2. Generative models with GAN
3. GAN architectures
    1. Basics
    2. LaPGAN
    3. DCGAN
    4. ProGAN
    5. MSG-GAN
    6. **StyleGAN**

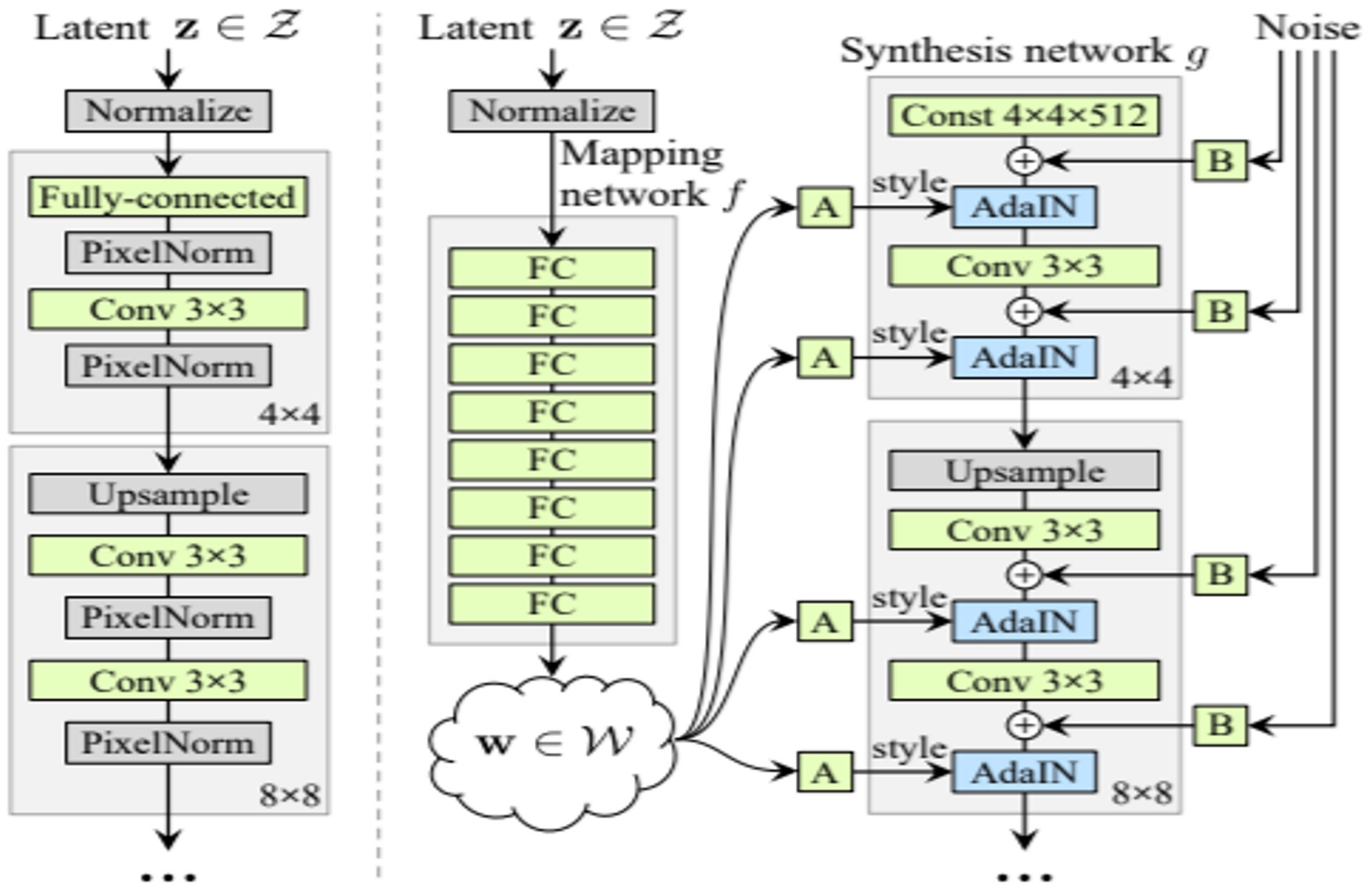
StyleGAN: A Style-Based Generator Architecture for Generative Adversarial Networks [Karras CVPR 2019]

Still progressive growing architecture but with new refinement block based on: Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization (AdaIN)

$$\mathrm{AdaIN}(x, y) = \sigma(y)\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \mu(y)$$



StyleGAN Network Architecture



(a) Traditional (b) Style-based generator

Building up the Model

	Method	CelebA-HQ	FFHQ
A	Baseline Progressive GAN [29]	7.79	8.04
B	+ Tuning (incl. bilinear up/down)	6.11	5.25
C	+ Add mapping and styles	5.34	4.85
D	+ Remove traditional input	5.07	4.88
E	+ Add noise inputs	**5.06**	4.42
F	+ Mixing regularization	5.17	**4.40**

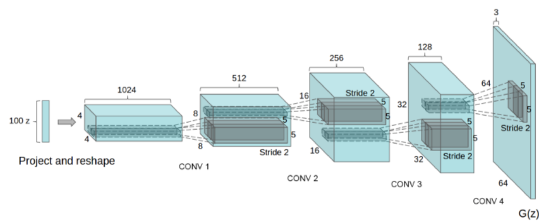
Generative models

Outline

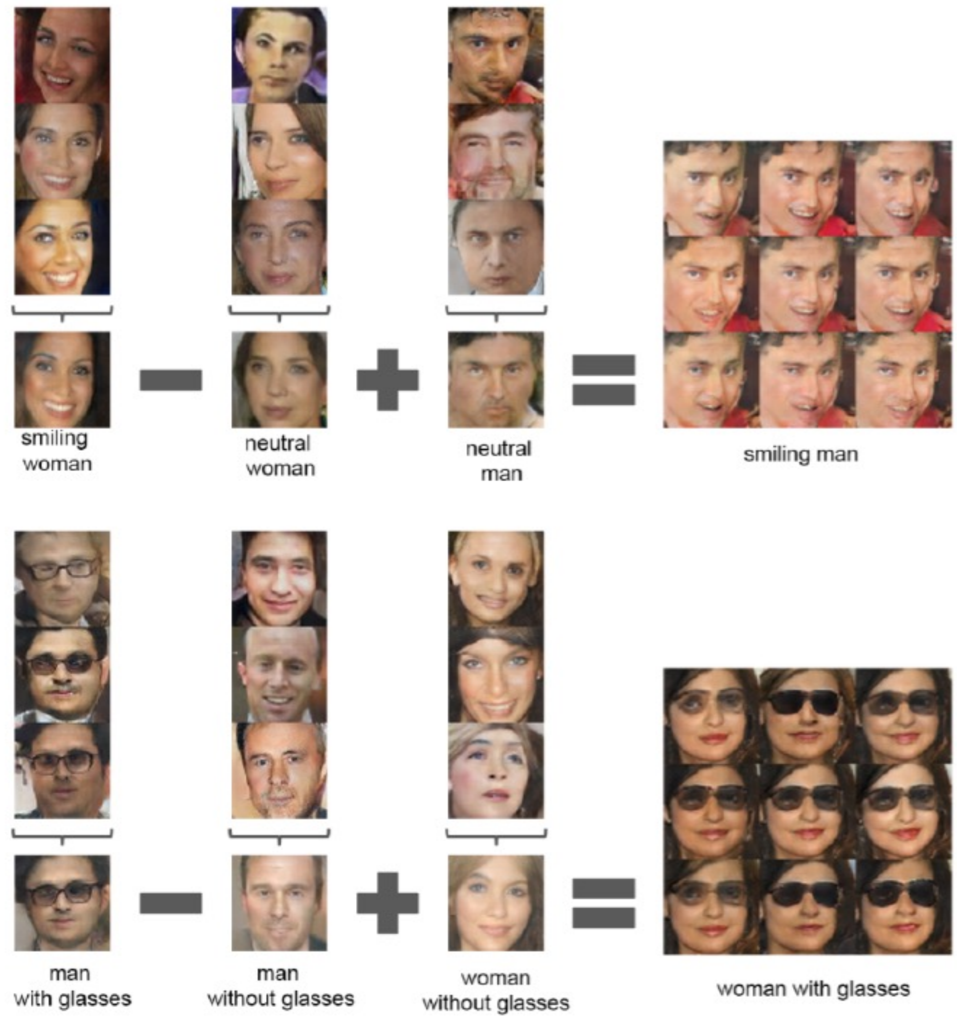
1. Preview: Auto-Encoders, VAE
2. Generative models with GAN
3. GAN architectures
4. **Editing**

DCGAN: Arithmetics in z space

Latent space analysis for GAN editing



Artithmetics in latent space



DCGAN Walking in latent

Gan Editing

Latent space analysis for GAN editing