GENERATIVE ADVERSARIAL NETWORKS

Sergey Nikolenko

Harbour Space University

June 23, 2026

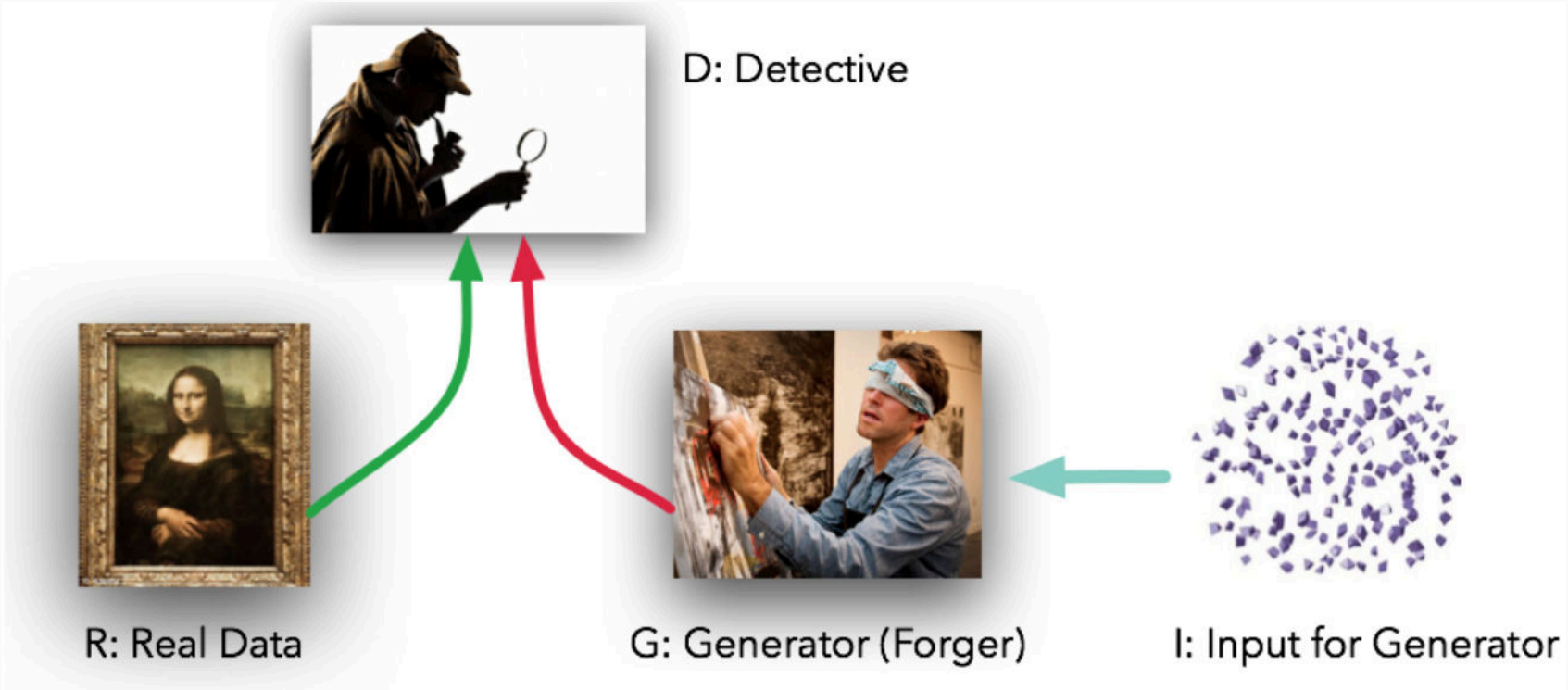
Random facts:

- on June 23, 1314, the Battle of Bannockburn began near Stirling, where Robert the Bruce's heavily outnumbered Scots routed the much larger English army of Edward II and secured Scottish independence for a generation
- on June 23, 1868, Christopher Latham Sholes patented the "Type-Writer"; to stop the mechanical type-bars from jamming, he scattered the letters into the deliberately awkward QWERTY layout we are all still typing on a century and a half later
- on June 23, 1894, Baron Pierre de Coubertin convened a congress at the Sorbonne that founded the International Olympic Committee and resolved to revive the Olympic Games; the date is celebrated every year since as Olympic Day
- on June 23, 1926, the College Board administered the first SAT exam, still a landmark for US students
- on June 23, 1991, Sega released *Sonic the Hedgehog* in North America; the blue mascot was engineered to be faster and edgier than Nintendo's Mario, and for a few years it actually pushed Sega ahead of Nintendo in the U.S. console war

Generative adversarial networks

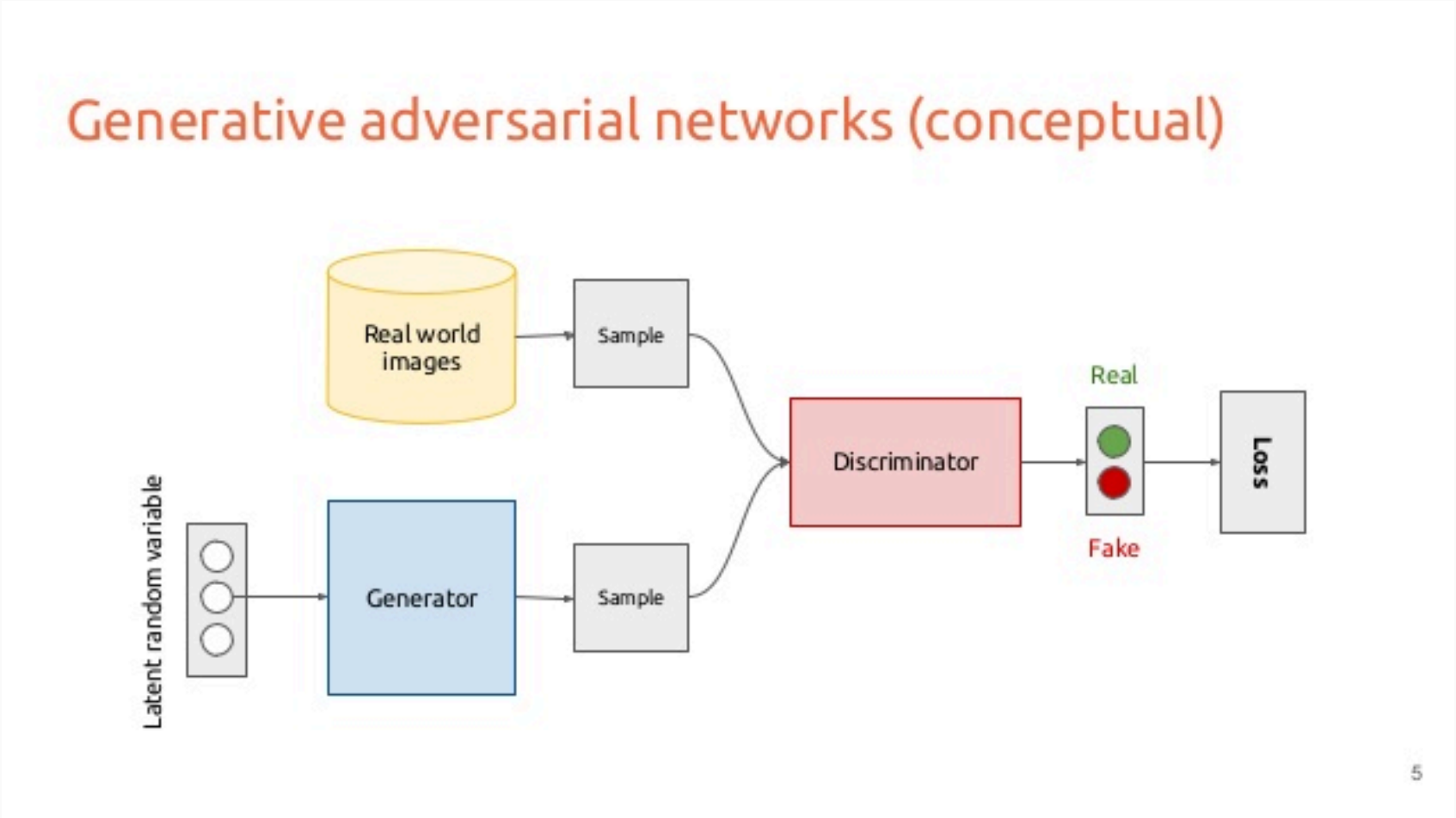
GAN

- Generative adversarial networks (GAN): generator vs. discriminator, p_{data} vs. p_g.



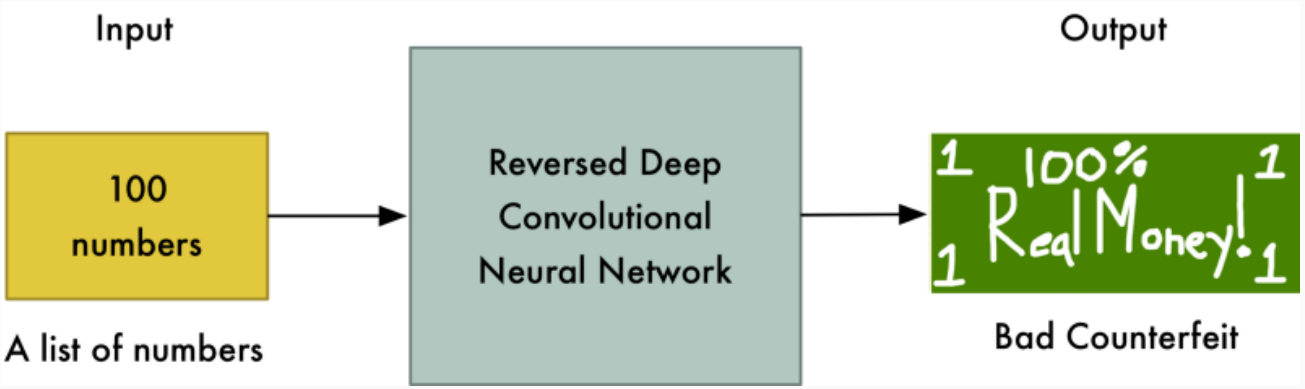
GAN

- Generative adversarial networks (GAN): generator vs. discriminator, p_{data} vs. p_g.

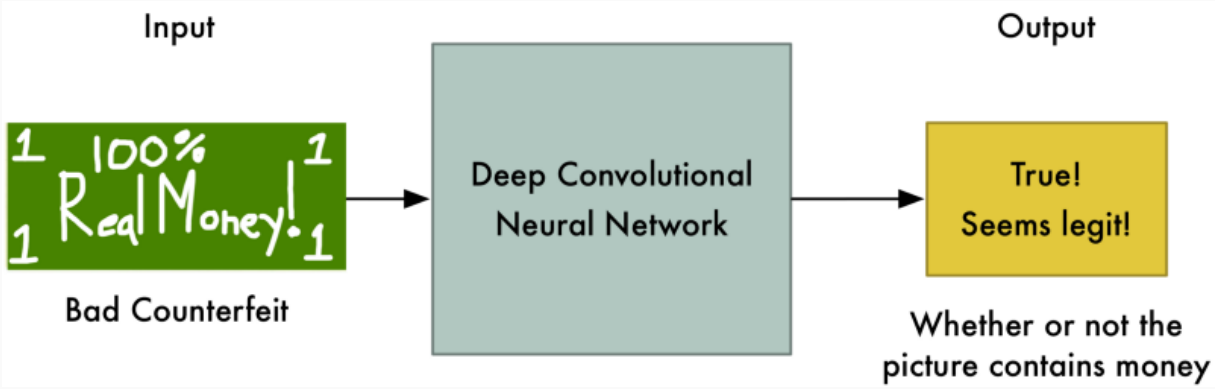


GAN

- Ideally it should look roughly like this (Geitgey, 2017):
  - at first the generator is bad:

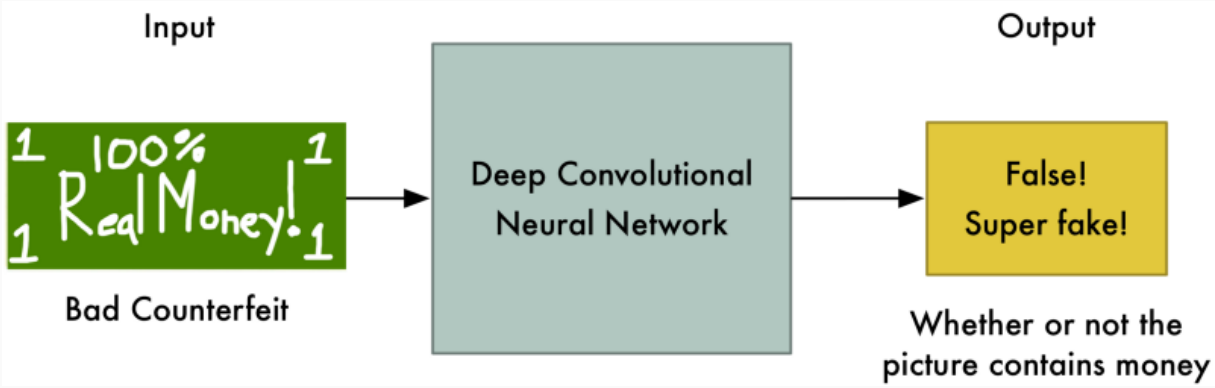


  - but the discriminator can't do anything either:

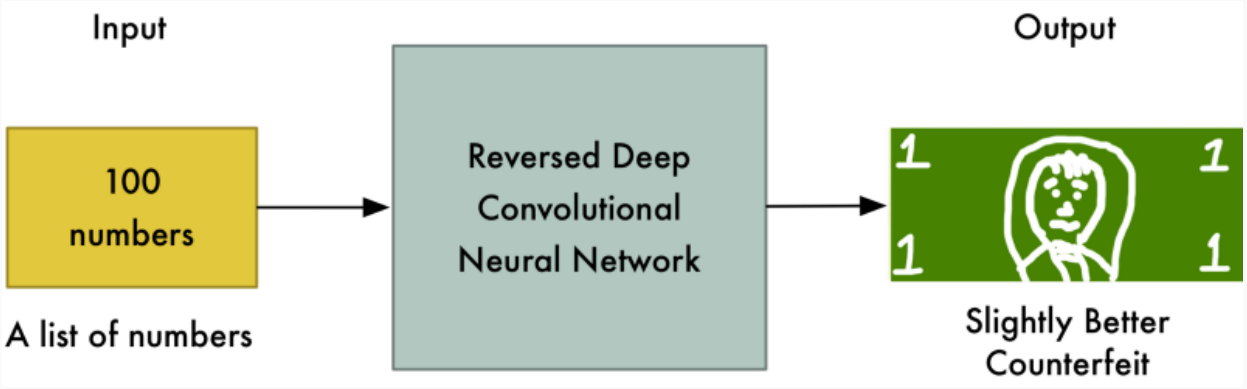


GAN

- Ideally it should look roughly like this (Geitgey, 2017):
  - so first we train the discriminator a little:



  - ...and the generator will have to come up with something better:



- And they will keep going like this until everything works out.

GAN

- Formally, the generator is a function

$$G = G(\mathbf{z}; \theta_g) : Z \to X,$$

and the discriminator is a function

$$D = D(\mathbf{x}; \theta_d) : X \to [0, 1].$$

- The objective function for the discriminator is binary classification:

$$\mathrm{E}_{\mathbf{x} \sim p_{\mathrm{data}}(\mathbf{x})} \left[\log D(\mathbf{x}) \right] + \mathrm{E}_{\mathbf{x} \sim p_g(\mathbf{x})} \left[\log(1 - D(\mathbf{x})) \right],$$

where $p_g(\mathbf{x}) = G_{\mathbf{z} \sim p_z}(\mathbf{z})$ is the distribution generated by G; that is, we take two mini-batches, with labels 0 from the generator and labels 1 from the data.

GAN

- And the generator learns to fool the discriminator, that is, to minimize

$$\mathrm{E}_{\mathbf{x}\sim p_g(\mathbf{x})}\left[\log(1-D(\mathbf{x}))\right] = \mathrm{E}_{\mathbf{z}\sim p_z(\mathbf{z})}\left[\log(1-D(G(\mathbf{z})))\right].$$

- And now, if we combine them, we get a typical minimax game:

$$\min_G \max_D V(D,G), \text{ where}$$

$$V(D,G) = \mathrm{E}_{\mathbf{x}\sim p_{\mathrm{data}}(\mathbf{x})}[\log D(\mathbf{x})] + \mathrm{E}_{z\sim p_z(z)}[\log(1-D(G(z)))].$$

GAN

- One can prove that $\max_D V(D, G)$ is minimized exactly when $p_g = p_{\text{data}}$.
- An important property: for a fixed generator G, the optimal discriminator distribution D — is the distribution

$$D^*_G(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})}.$$

- (Exercise: try to prove this)

GAN

- The global minimum of the criterion is attained if and only if $p_g = p_{\text{data}}$; the criterion itself for the optimal D is equivalent to

$$\text{KL}\left(p_{\text{data}} \middle\| \frac{p_{\text{data}} + p_g}{2}\right) + \text{KL}\left(p_g \middle\| \frac{p_{\text{data}} + p_g}{2}\right),$$

  this is the so-called *Jensen–Shannon divergence*.
- But these are all results for the generator loss $\text{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$.
- And this is not the best loss function for the generator...

GAN

- What is wrong with minimizing $\mathrm{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$?
  - cross-entropy is good in a classifier: if the answer is wrong, it does not saturate, and if it is right, it saturates and does not change;
  - here it turns out that when the discriminator works well, its gradient vanishes...
  - ...and the generator gradient vanishes too!
- That is why in practice we minimize

$$-\mathrm{E}_{z \sim p_z(z)}[\log D(G(\mathbf{z}))],$$

  i.e. we maximize the probability that D made a mistake, rather than minimizing the probability of its correct answer.

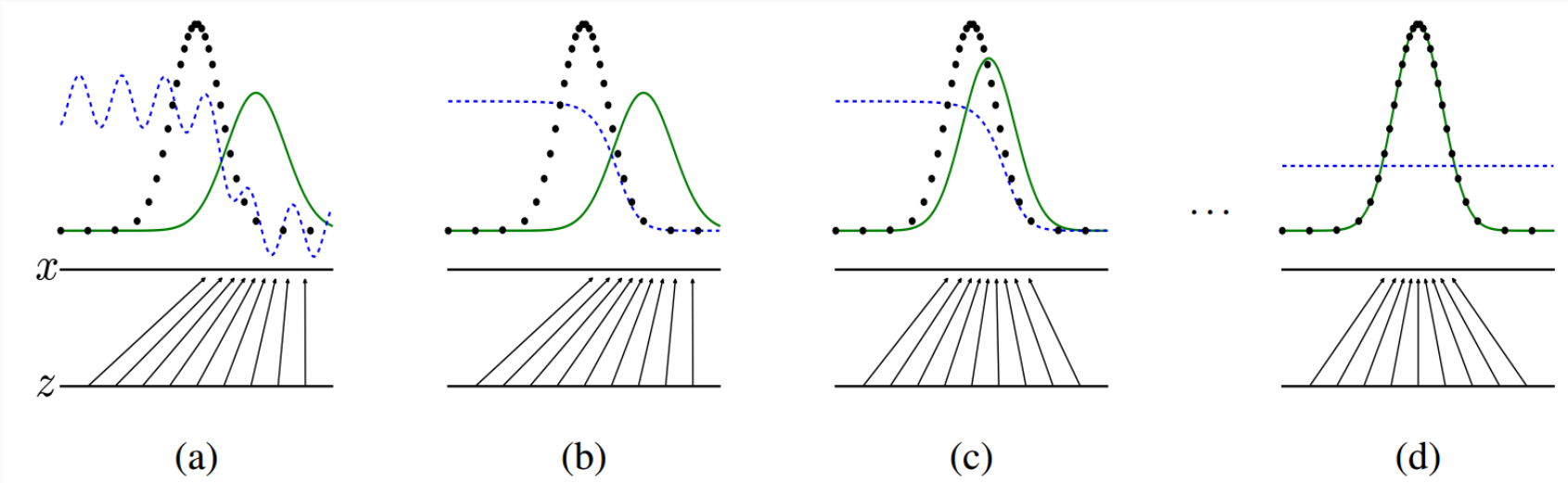
GAN

- Training is essentially similar to the EM algorithm – alternating:
  - we fix G, update the discriminator with the loss function

$$\mathrm{E}_{\mathbf{x}\sim p_{\mathrm{data}}(\mathbf{x})}\left[\log D(\mathbf{x})\right] + \mathrm{E}_{\mathbf{x}\sim p_g(\mathbf{x})}\left[\log(1 - D(\mathbf{x}))\right],$$

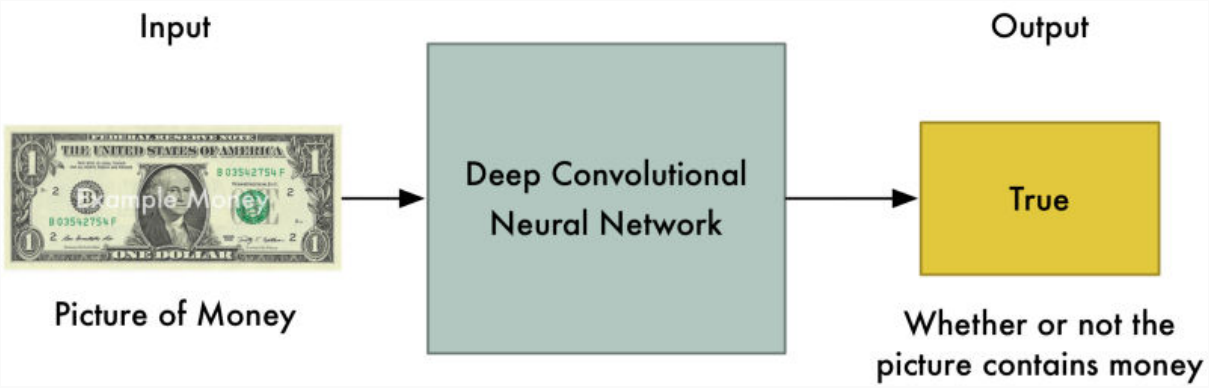
  - we fix D, update the generator with the loss function

$$-\mathrm{E}_{\mathbf{x}\sim p_{\mathrm{data}}(\mathbf{x})}[\log D(\mathbf{x})],$$

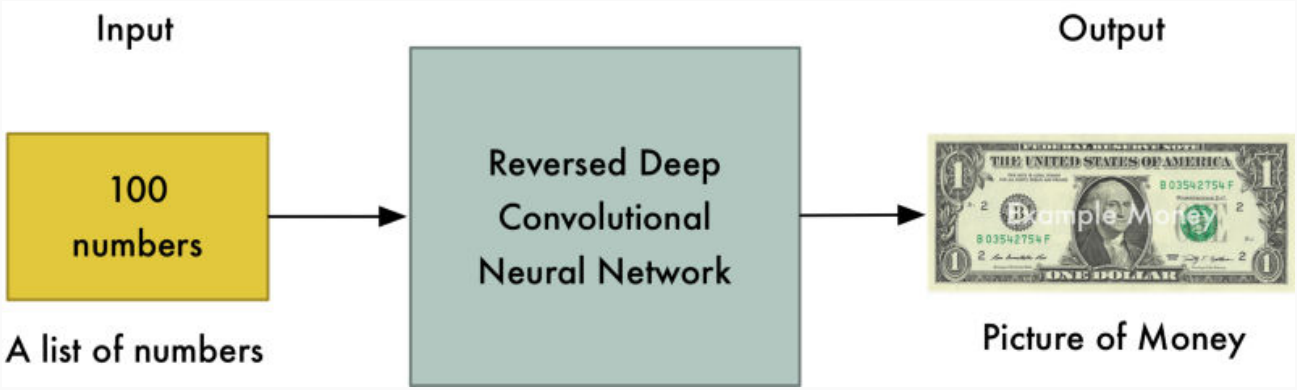


GAN

- Ideally it should look roughly like this (Geitgey, 2017):
  - the discriminator tries to recognize:

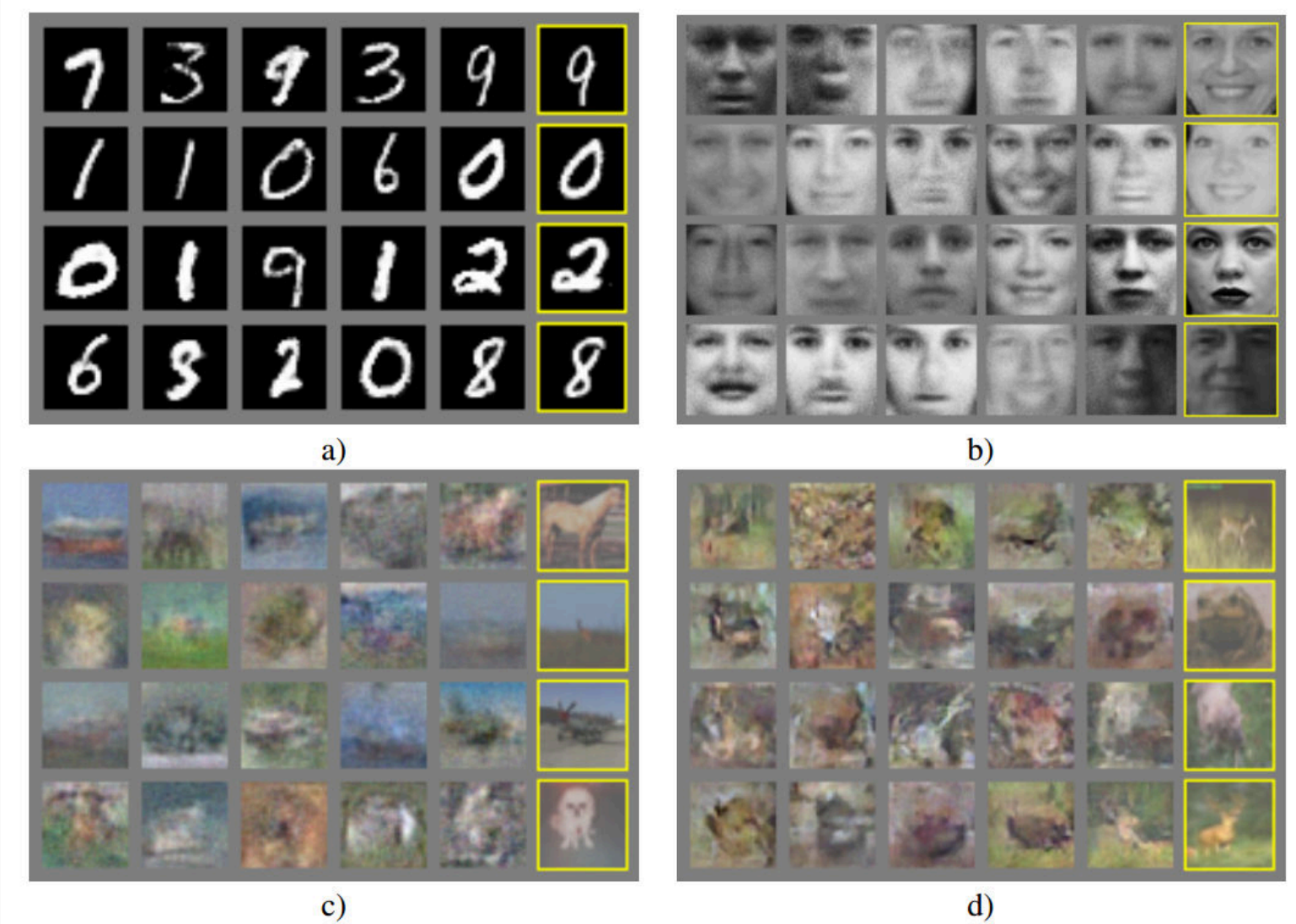


  - and the generator wants to generate:



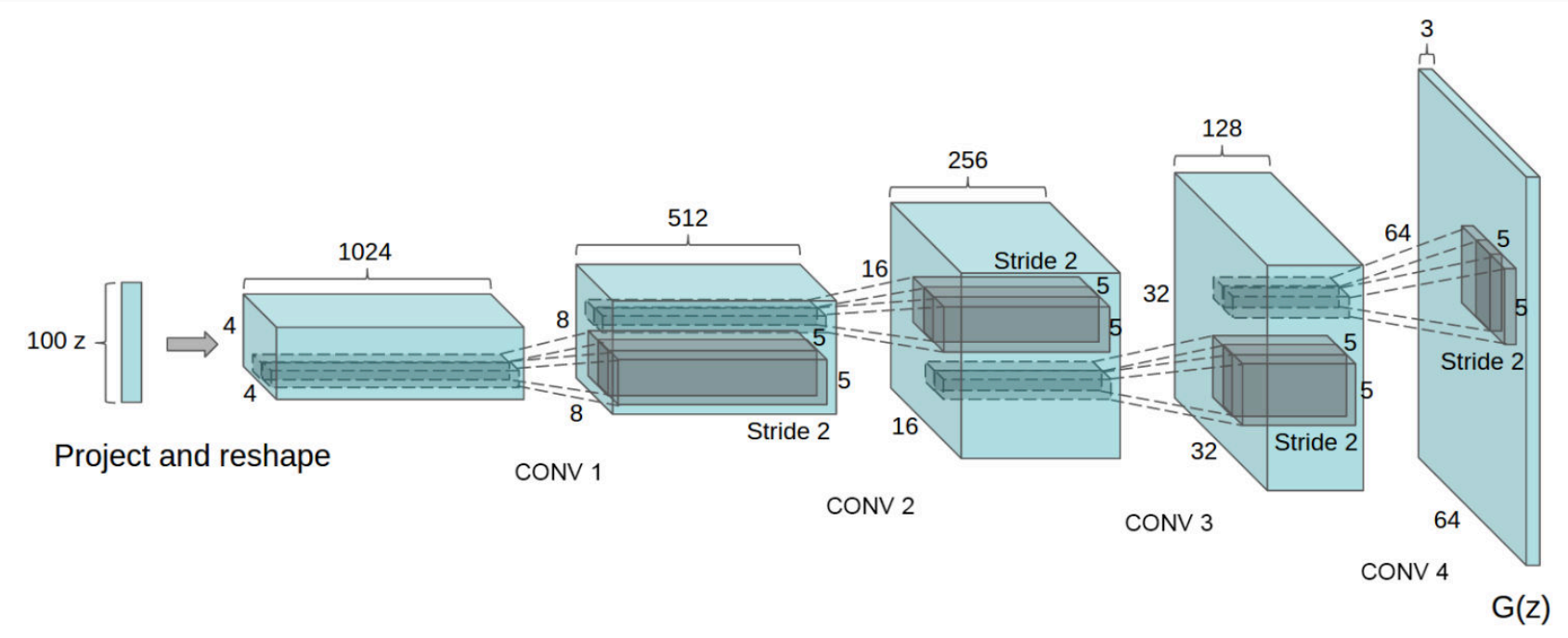
GAN

- Sampling from a GAN (Goodfellow et al., 2014):



DCGAN

- DCGAN – Deep Convolutional Generative Adversarial Networks (Radford et al., 2016).
- Several new tricks (strided convolutions instead of pooling, batchnorm everywhere, removing fully connected layers, Adam…)

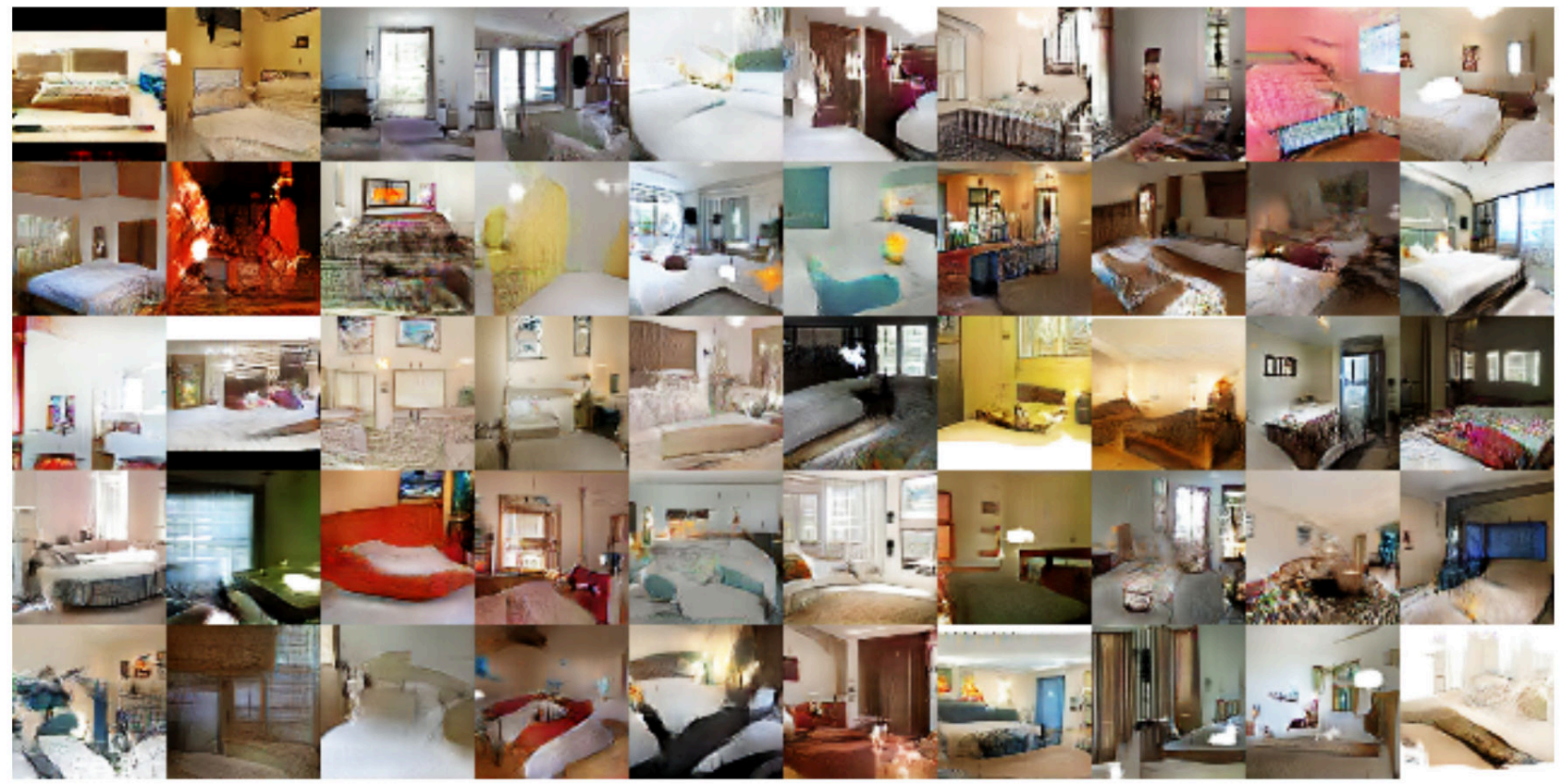


DCGAN

- Bedrooms (LSUN) after one epoch:

DCGAN

- Bedrooms (LSUN) after five epochs:



DCGAN

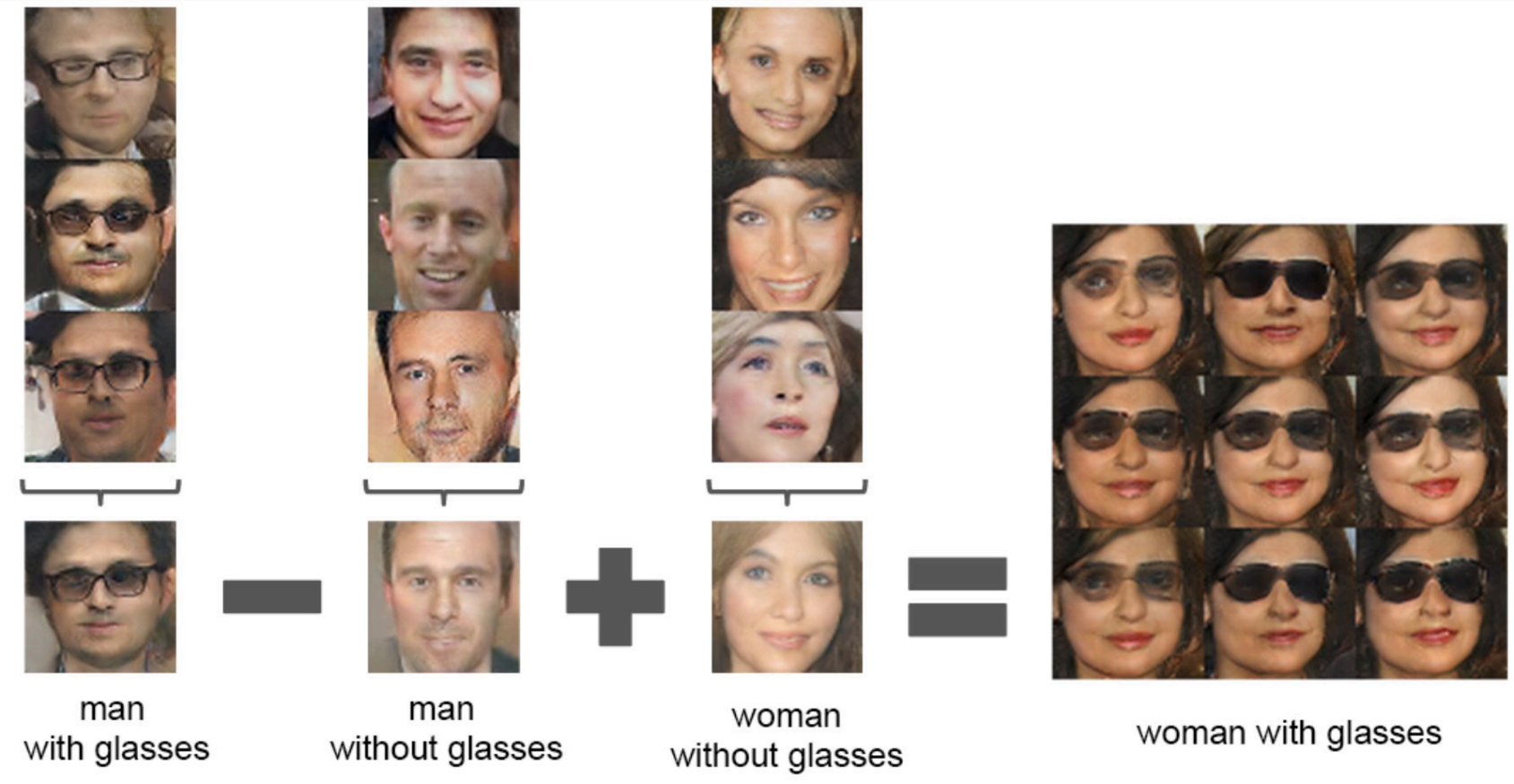
- A walk through feature space:

DCGAN

- Removing filters that detect windows:

DCGAN

- Vector arithmetic (as in *word2vec*):



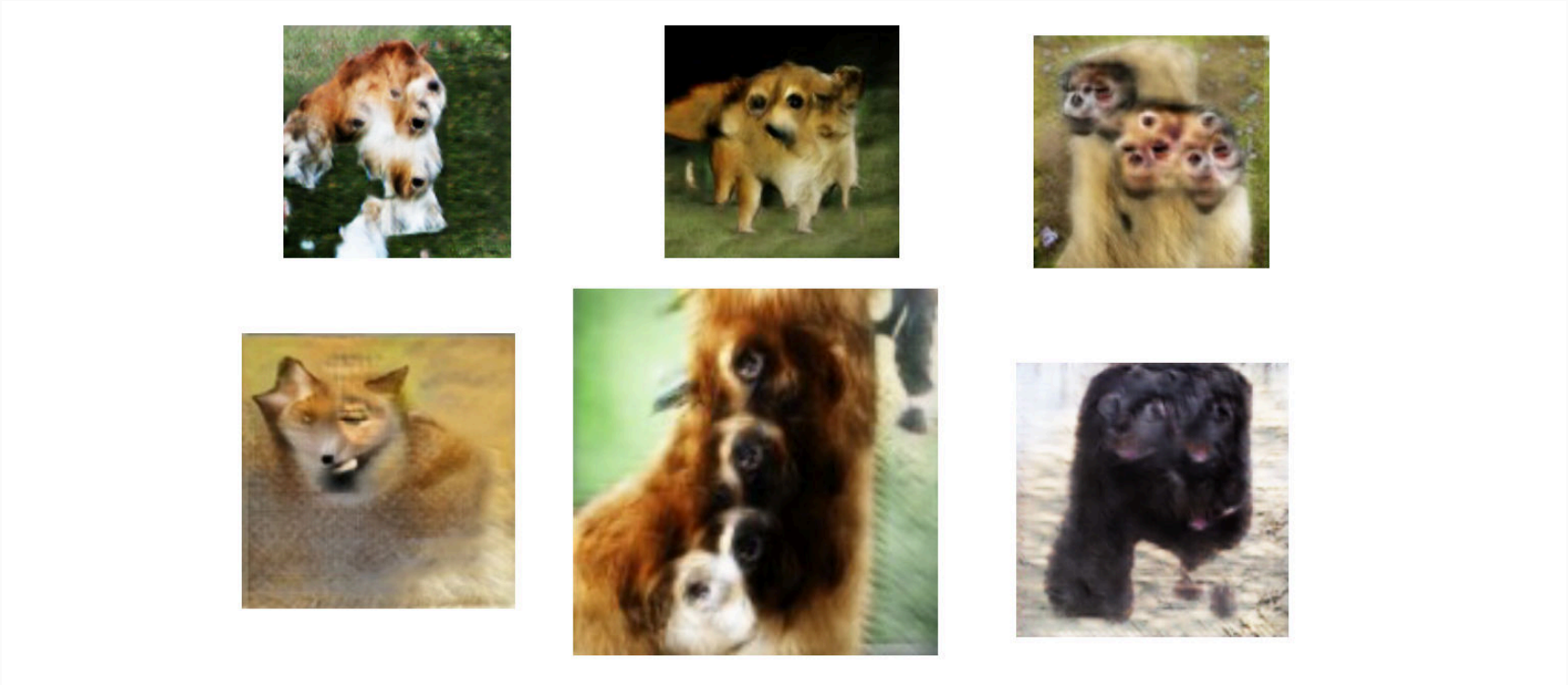
DCGAN

- (Geitgey, 2017): pixel art from NES games

- Animation: `https://medium.com/@ageitgey/abusing-generative-adversarial-networks-to-make-8-bit-pixel-art-e45d9b96cee7`

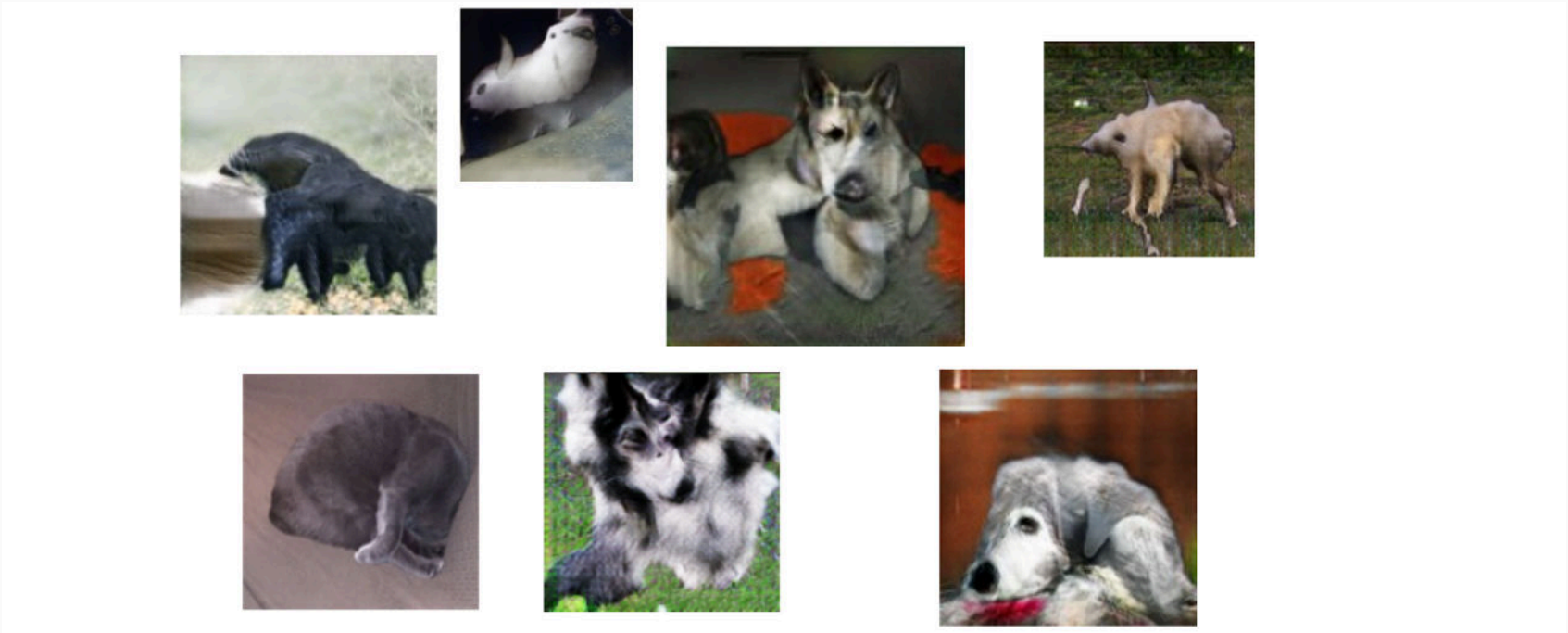
DCGAN

- But of course it does not always work – problems with counting:



DCGAN

- But of course it does not always work – problems with perspective (right?):



Progress

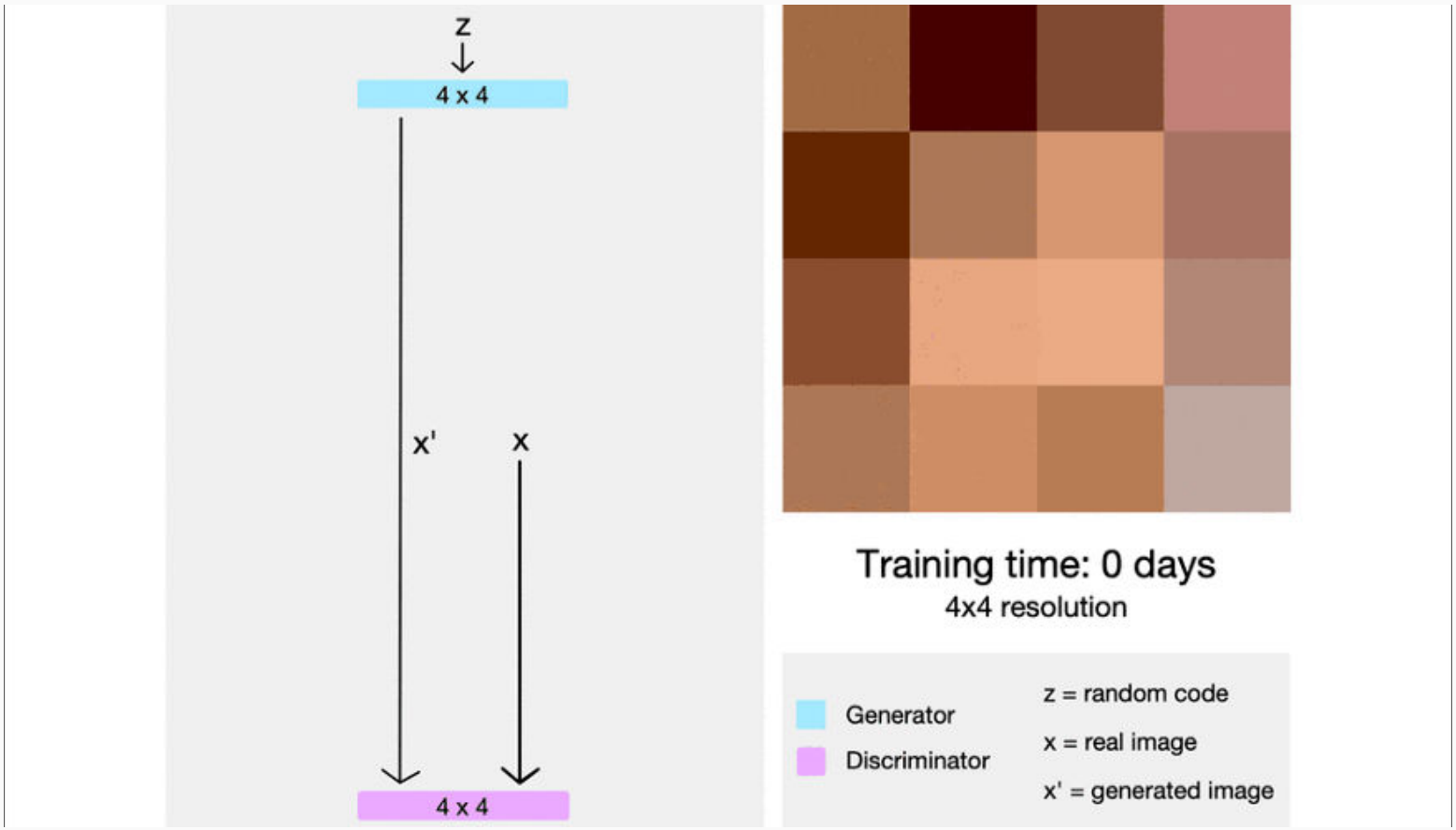
- This is how the very first GANs worked.
- But we have come a long way since then:

- How did that happen?

Enlarge your GAN

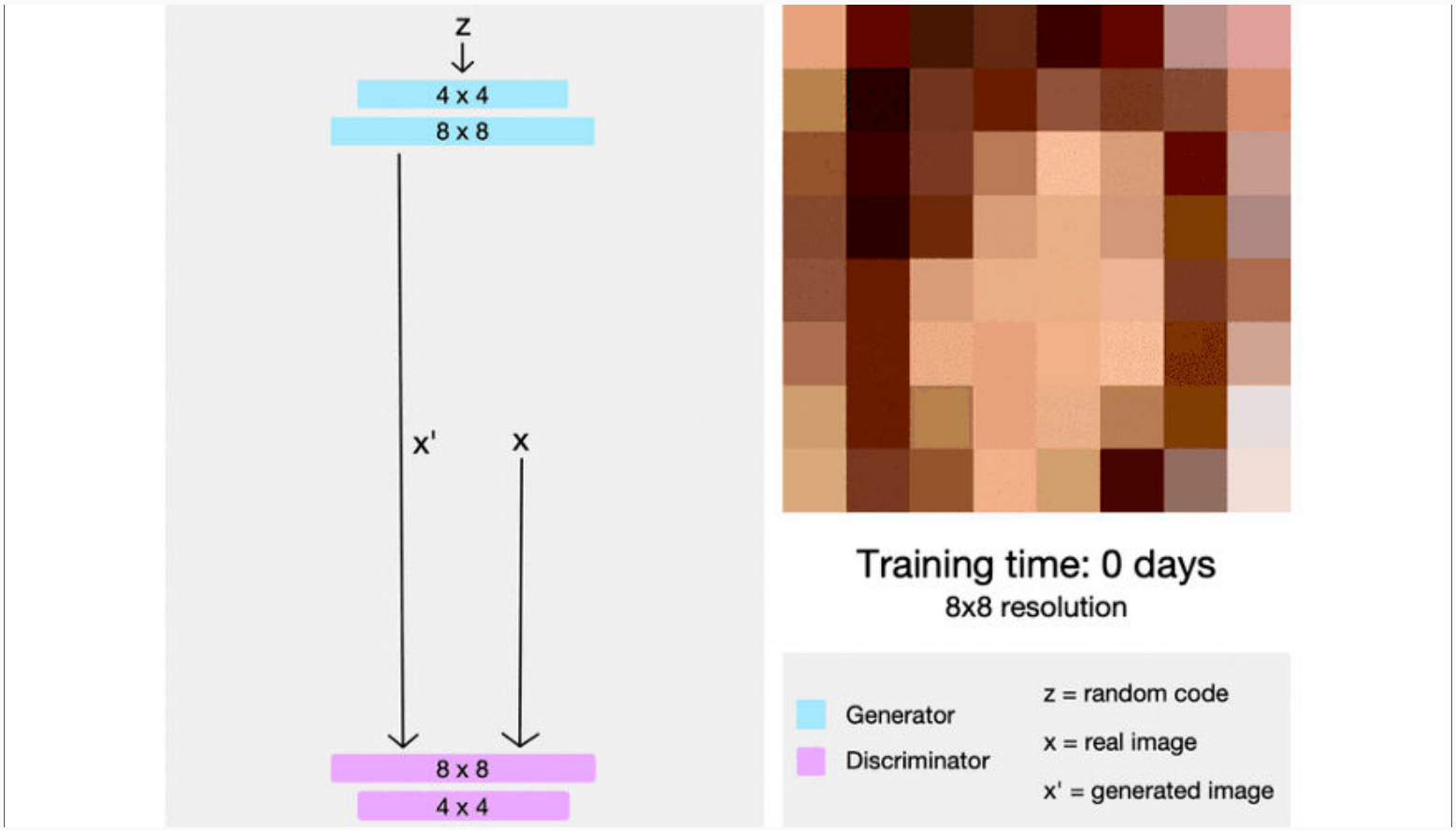
ProGAN

- An important breakthrough came from NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): at each stage we train the GAN to create images four times larger, starting from 4×4
- And this way they gradually increase the resolution up to 1024×1024



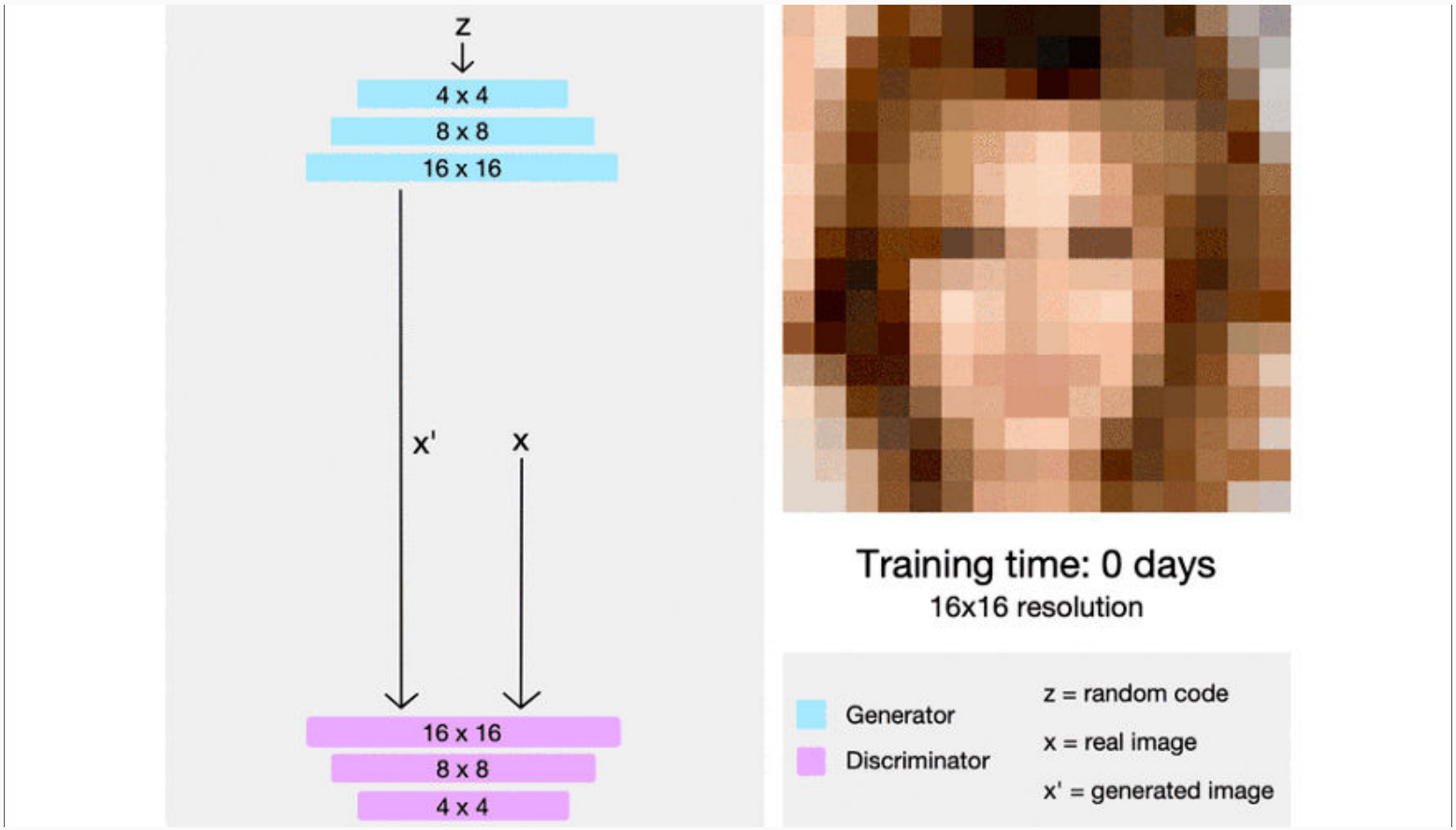
ProGAN

- An important breakthrough came from NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): at each stage we train the GAN to create images four times larger, starting from 4×4
- And this way they gradually increase the resolution up to 1024×1024



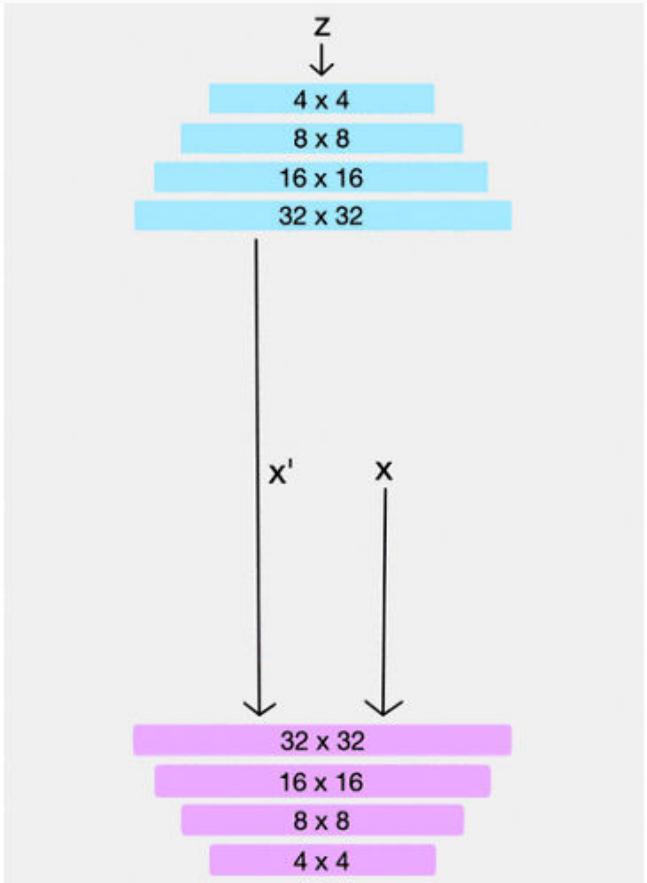
ProGAN

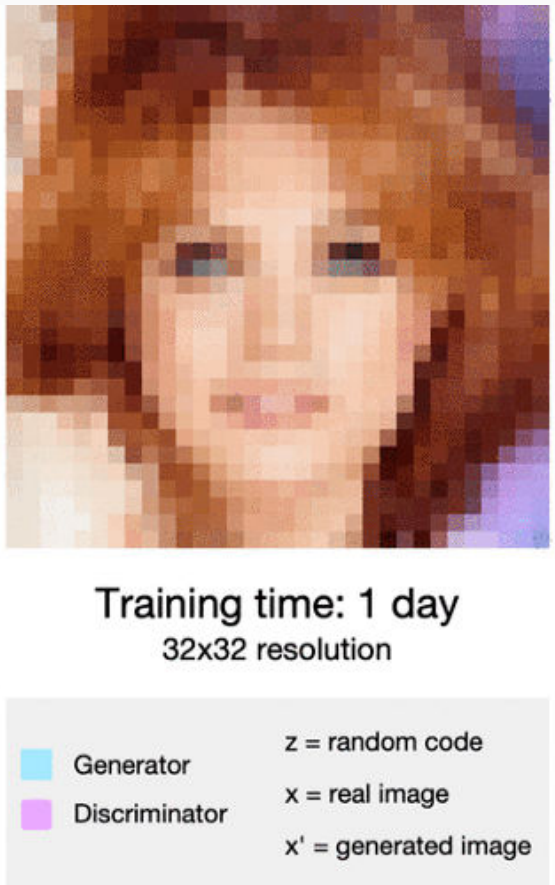
- An important breakthrough came from NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): at each stage we train the GAN to create images four times larger, starting from 4×4
- And this way they gradually increase the resolution up to 1024×1024



ProGAN

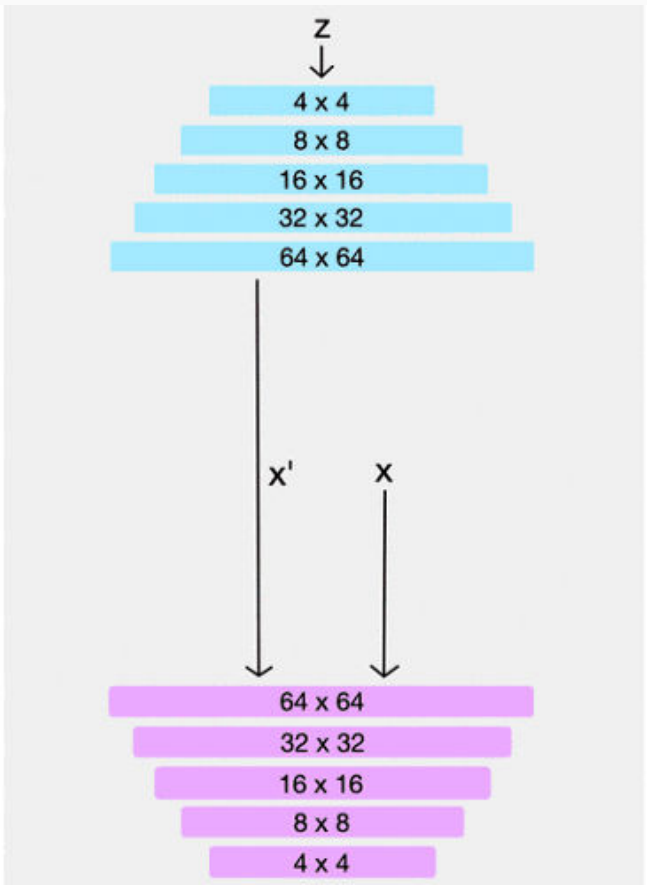
- An important breakthrough came from NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): at each stage we train the GAN to create images four times larger, starting from 4×4
- And this way they gradually increase the resolution up to 1024×1024

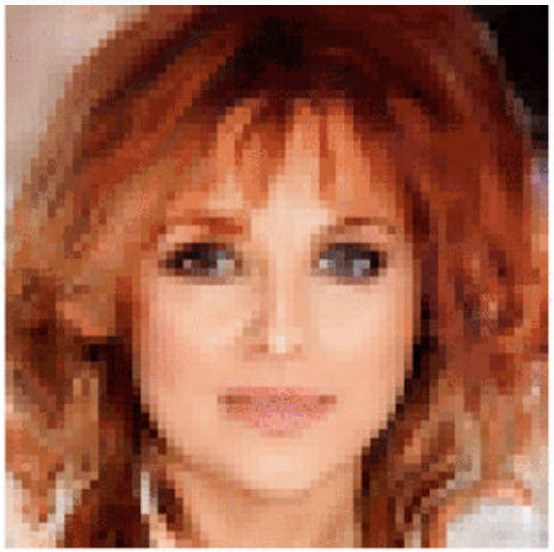




ProGAN

- An important breakthrough came from NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): at each stage we train the GAN to create images four times larger, starting from 4×4
- And this way they gradually increase the resolution up to 1024×1024





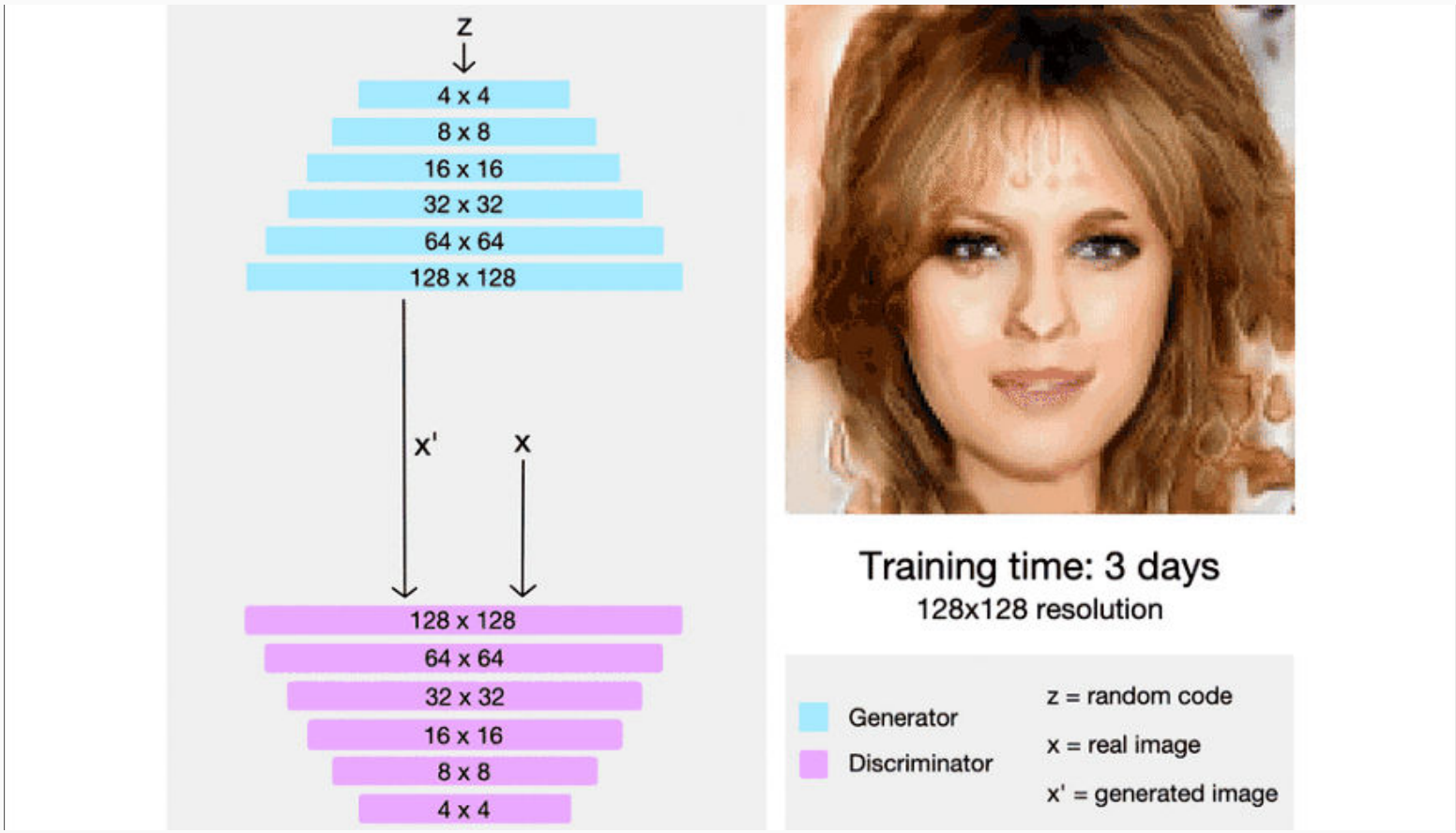
Training time: 2 days
64x64 resolution

Generator
Discriminator

z = random code
x = real image
x' = generated image

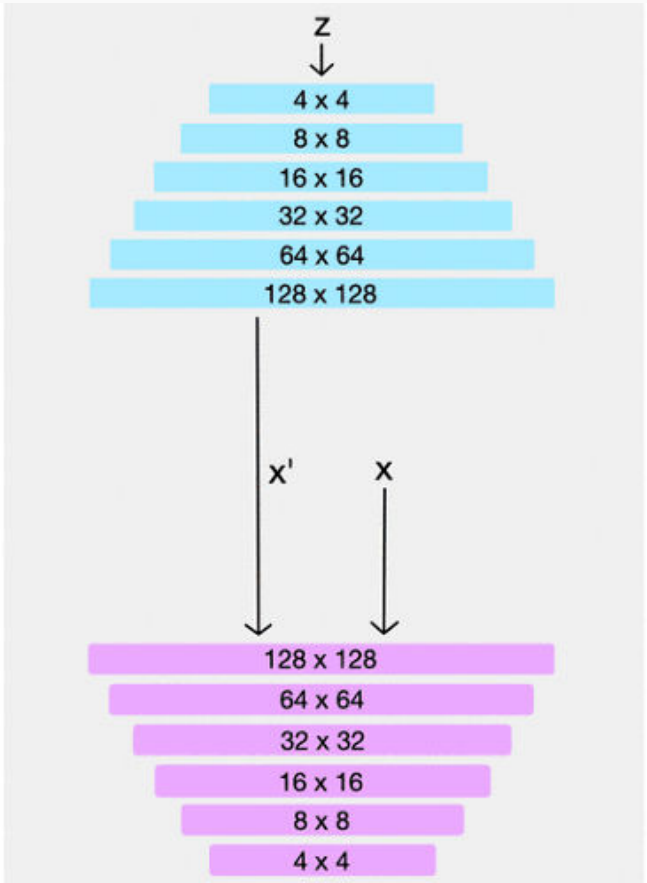
ProGAN

- An important breakthrough came from NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): at each stage we train the GAN to create images four times larger, starting from 4×4
- And this way they gradually increase the resolution up to 1024×1024



ProGAN

- An important breakthrough came from NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): at each stage we train the GAN to create images four times larger, starting from 4×4
- And this way they gradually increase the resolution up to 1024×1024



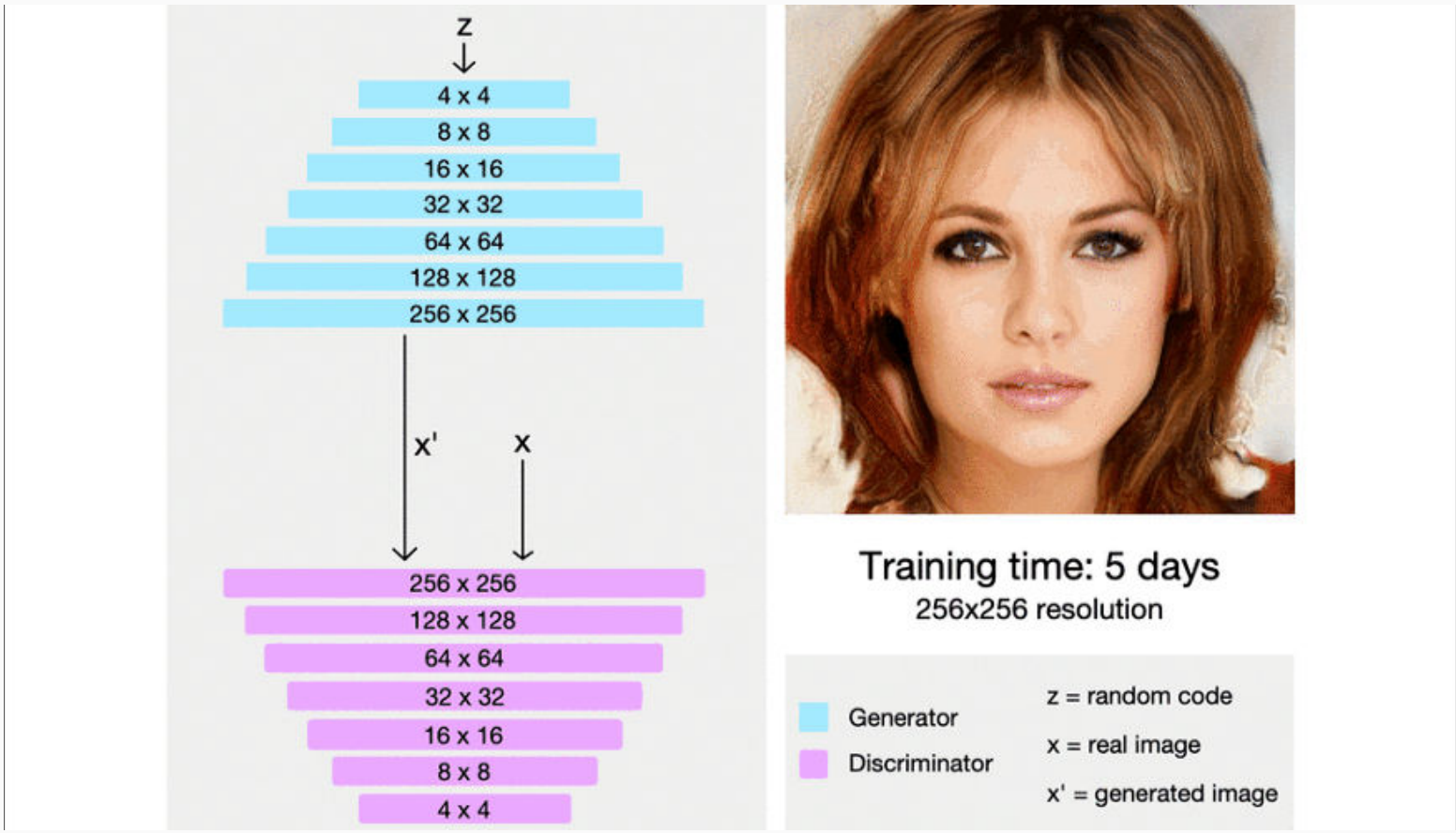
Training time: 4 days
128x128 resolution

Generator
Discriminator

z = random code
x = real image
x' = generated image

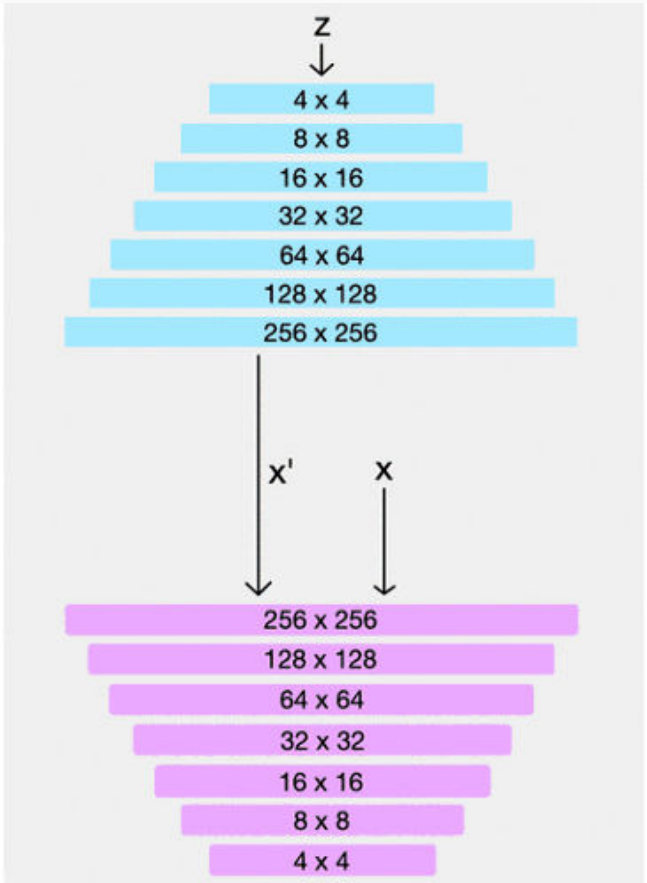
ProGAN

- An important breakthrough came from NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): at each stage we train the GAN to create images four times larger, starting from 4×4
- And this way they gradually increase the resolution up to 1024×1024



ProGAN

- An important breakthrough came from NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): at each stage we train the GAN to create images four times larger, starting from 4×4
- And this way they gradually increase the resolution up to 1024×1024



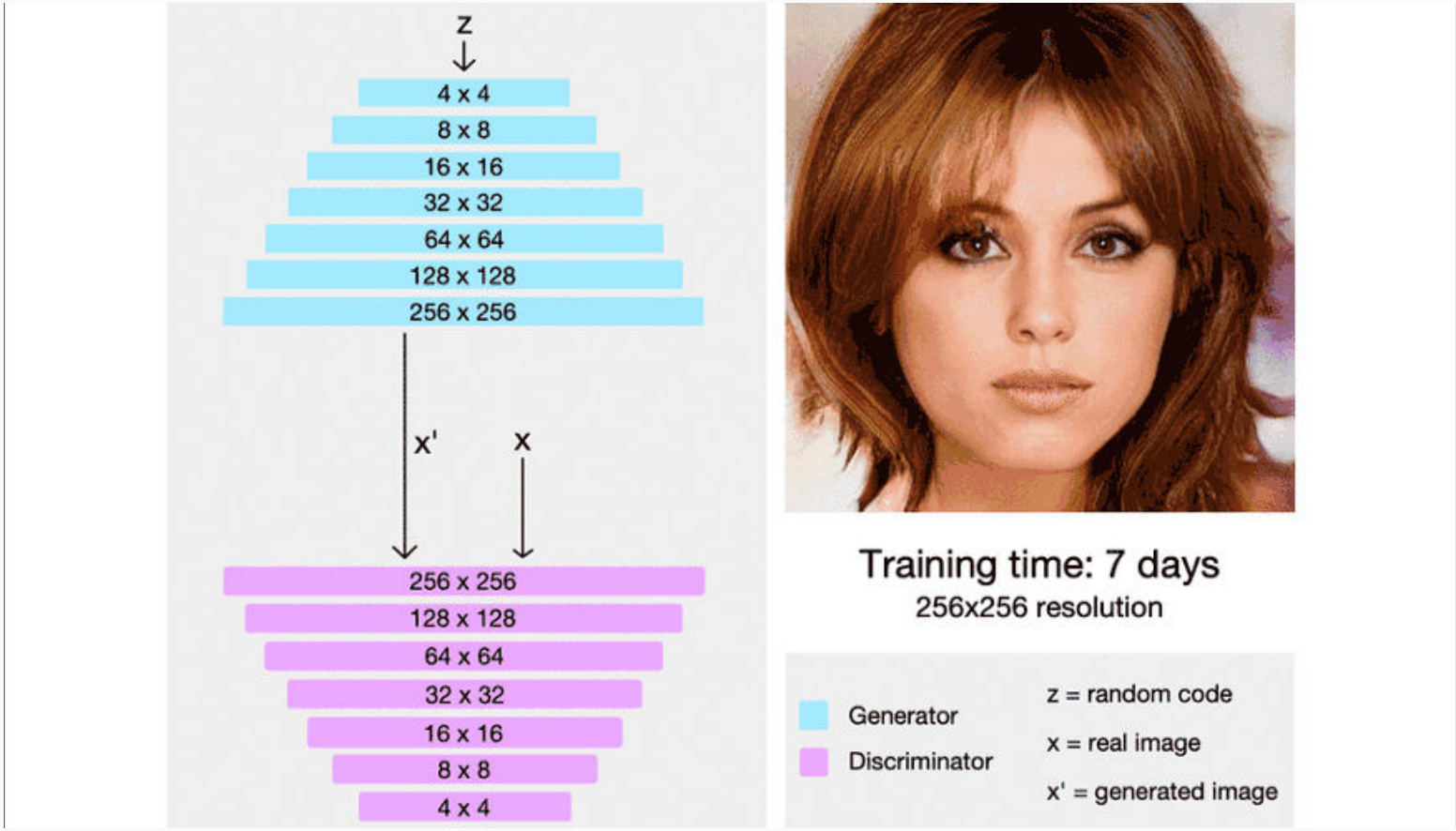
Training time: 6 days
256x256 resolution

Generator
Discriminator

z = random code
x = real image
x' = generated image

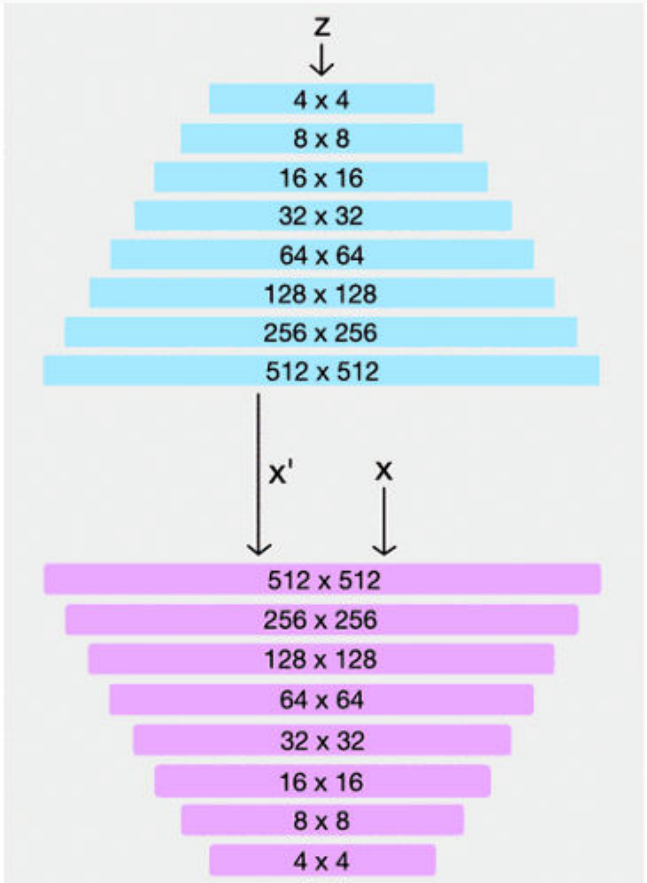
ProGAN

- An important breakthrough came from NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): at each stage we train the GAN to create images four times larger, starting from 4×4
- And this way they gradually increase the resolution up to 1024×1024



ProGAN

- An important breakthrough came from NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): at each stage we train the GAN to create images four times larger, starting from 4×4
- And this way they gradually increase the resolution up to 1024×1024



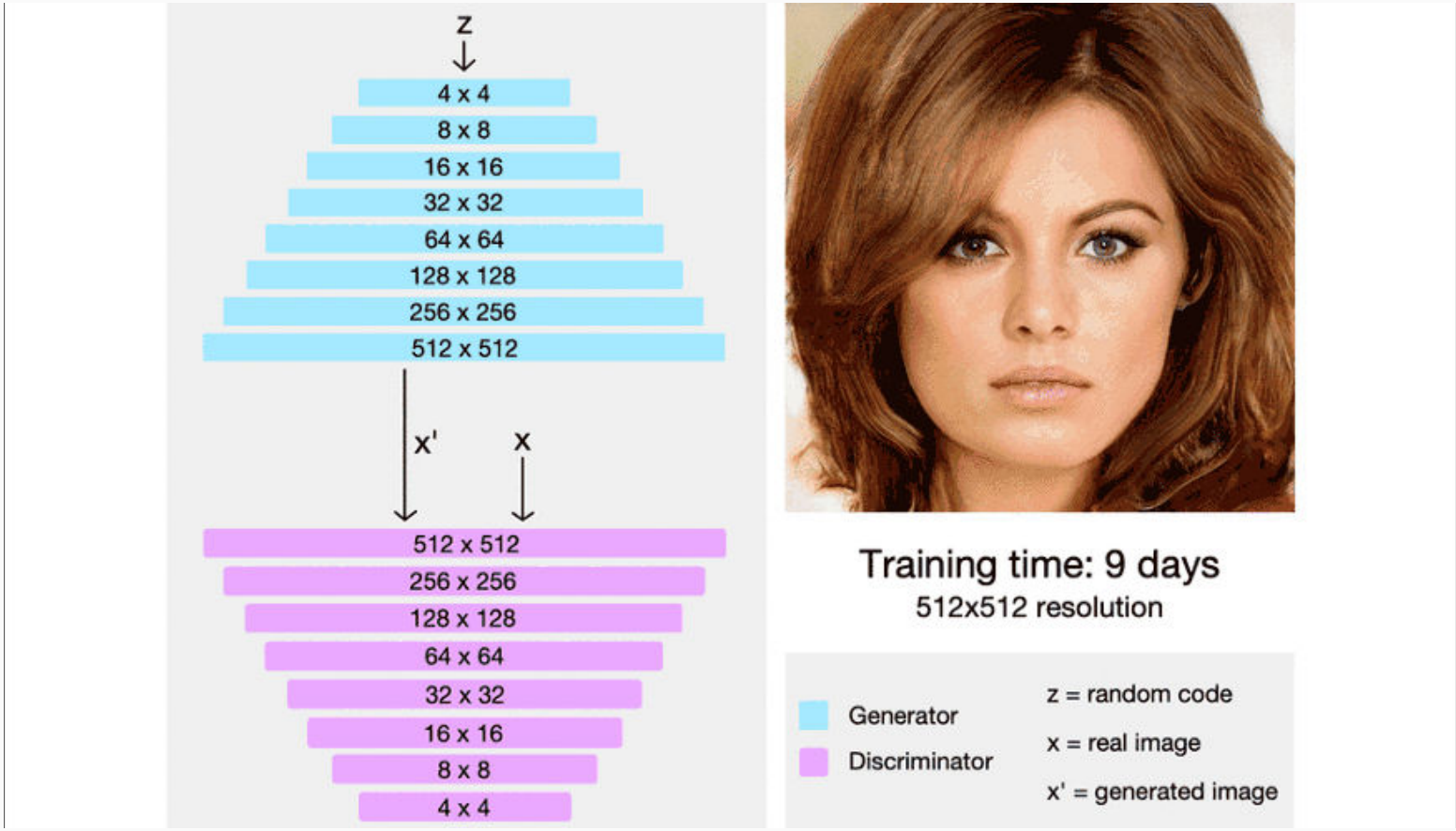
Training time: 8 days
512x512 resolution

Generator
Discriminator

z = random code
x = real image
x' = generated image

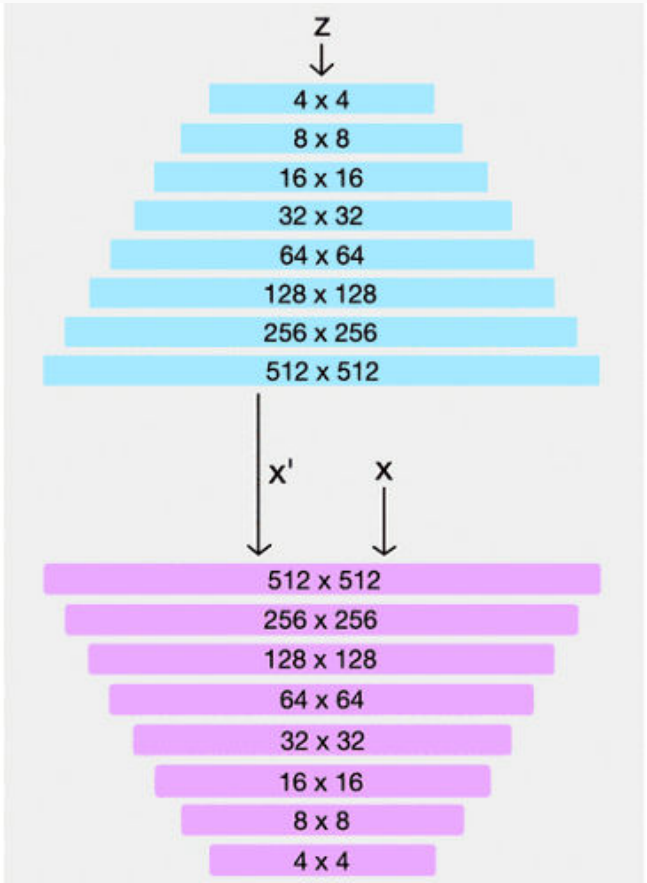
ProGAN

- An important breakthrough came from NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): at each stage we train the GAN to create images four times larger, starting from 4×4
- And this way they gradually increase the resolution up to 1024×1024



ProGAN

- An important breakthrough came from NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): at each stage we train the GAN to create images four times larger, starting from 4×4
- And this way they gradually increase the resolution up to 1024×1024



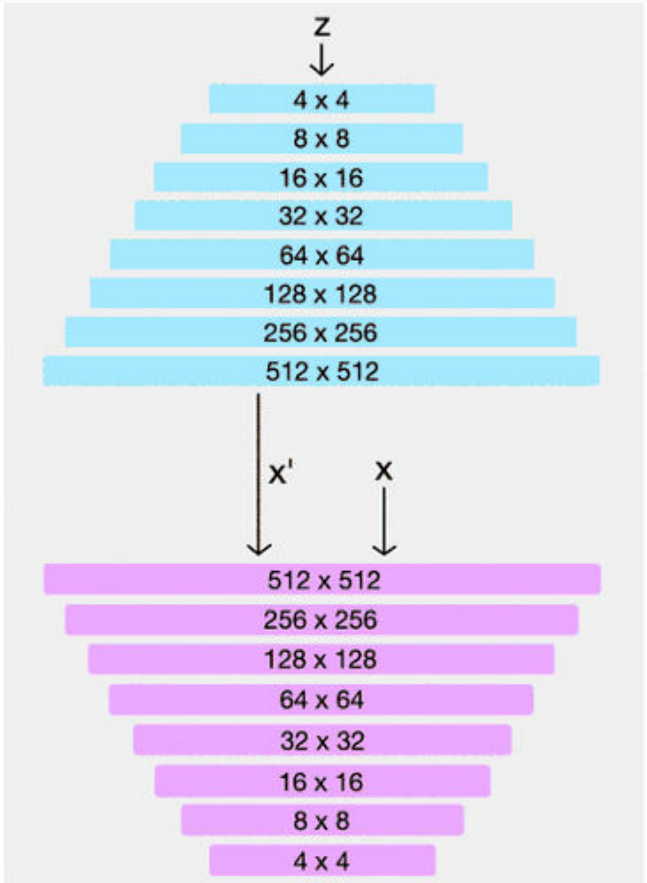
Training time: 10 days
512x512 resolution

Generator
Discriminator

z = random code
x = real image
x' = generated image

ProGAN

- An important breakthrough came from NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): at each stage we train the GAN to create images four times larger, starting from 4×4
- And this way they gradually increase the resolution up to 1024×1024



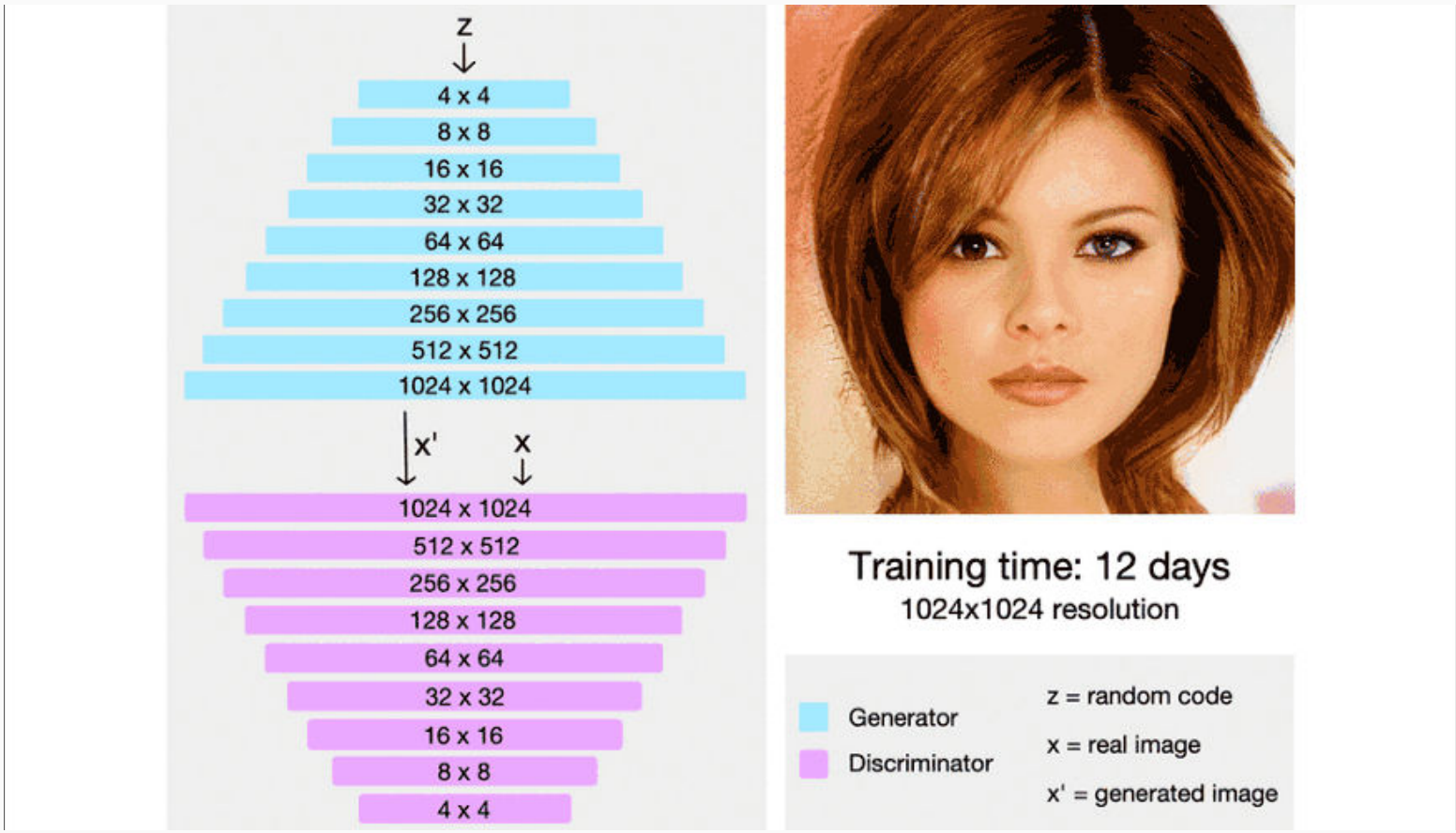
Training time: 11 days
512x512 resolution

Generator
Discriminator

z = random code
x = real image
x' = generated image

ProGAN

- An important breakthrough came from NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): at each stage we train the GAN to create images four times larger, starting from 4×4
- And this way they gradually increase the resolution up to 1024×1024

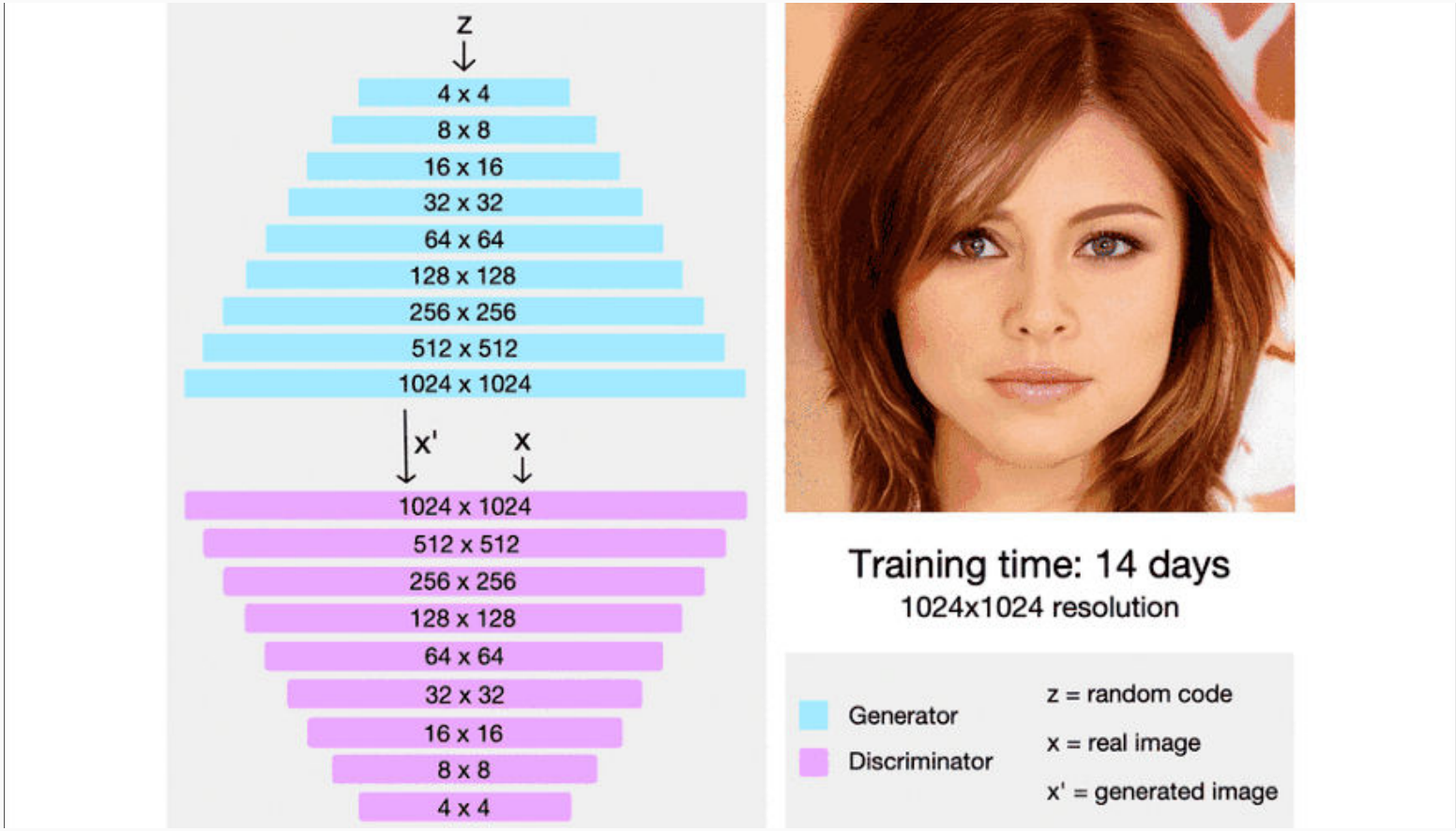


ProGAN

- An important breakthrough came from NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): at each stage we train the GAN to create images four times larger, starting from 4×4
- And this way they gradually increase the resolution up to 1024×1024

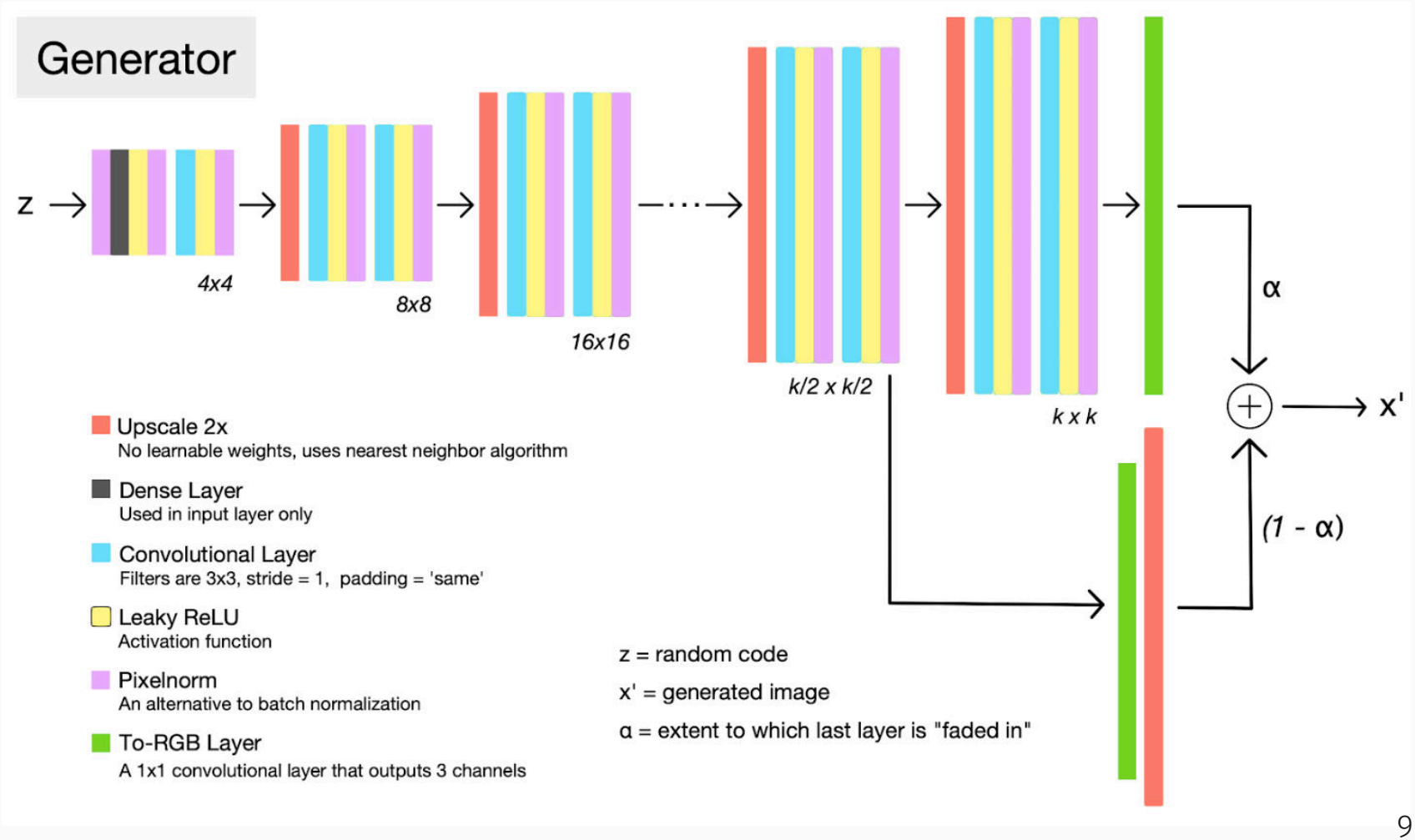
ProGAN

- An important breakthrough came from NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): at each stage we train the GAN to create images four times larger, starting from 4×4
- And this way they gradually increase the resolution up to 1024×1024



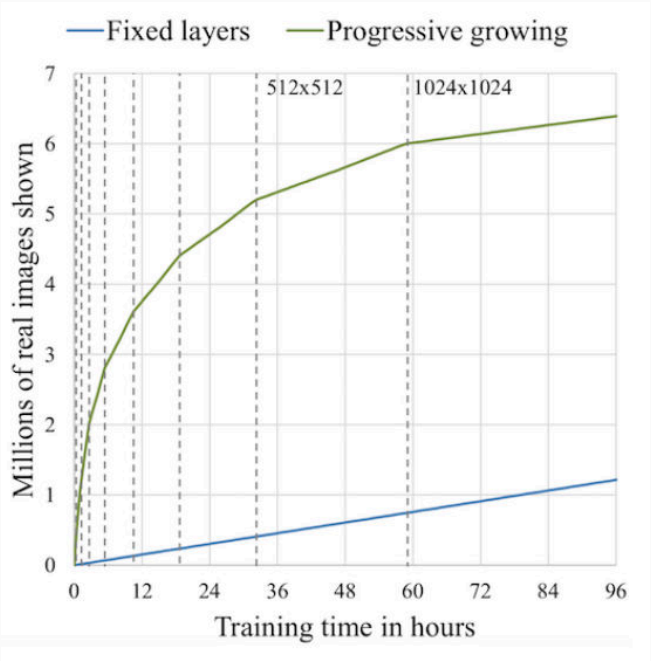
ProGAN

- Specifically, we add new layers each time; the discriminator is roughly symmetric



ProGAN

- This helps save training time, showing more data in the same time



- Let's take a look:
  - `https://www.youtube.com/watch?v=G06dEcZ-QTg`
  - `https://www.youtube.com/watch?v=36lE9tV9vm0`
- For other categories they did not go all the way to 1024×1024, but 256×256 looks pretty good

ProGAN

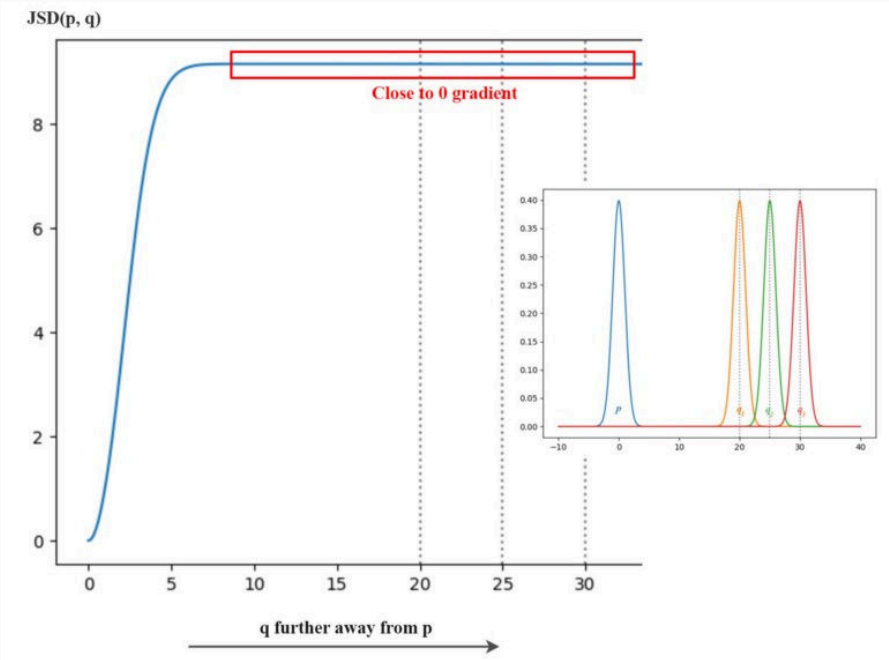
- For other categories they did not go all the way to 1024×1024, but 256×256 looks pretty good

- And all of this is trained in a fairly standard way: the authors compare the loss functions of least squares GAN and Wasserstein GAN...
- ...wait, what??? Let's figure it out...

Loss functions in GANs

LSGAN

- (Mao et al., 2016): Least Squares GAN (LSGAN)
- The problem with ordinary GANs: the loss function (Jensen-Shannon divergence) saturates when the generator distribution is far from the correct one, and nothing gets trained.



- This is because the discriminator in an ordinary GAN works like a classifier with a logistic sigmoid.

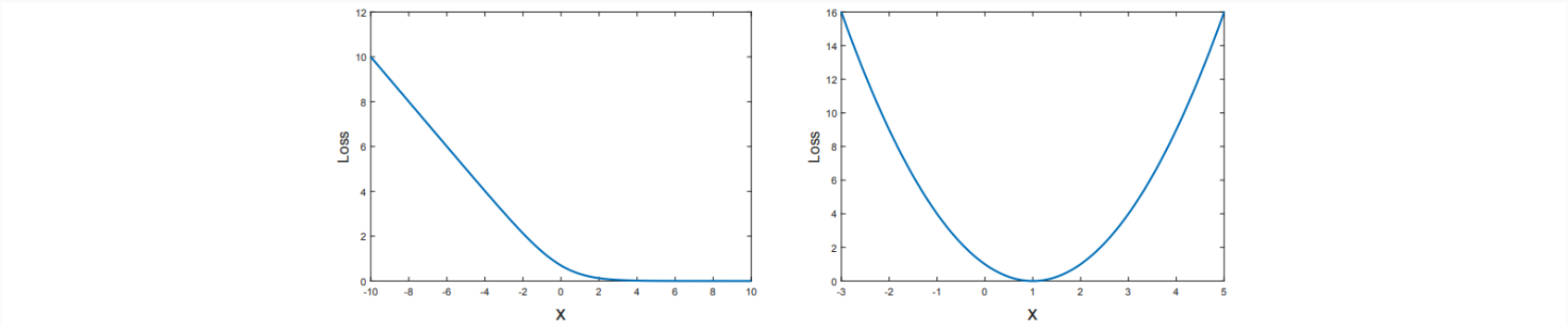
LSGAN

- Let's try switching from a sigmoidal to a quadratic loss function:

$$\min_D V_{\mathrm{LSGAN}}(D) = \frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}}\left[(D(\mathbf{x})-b)^2\right] + \frac{1}{2}\mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}}\left[(D(G(\mathbf{z}))-a)^2\right],$$

$$\min_G V_{\mathrm{LSGAN}}(G) = \frac{1}{2}\mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}}\left[(D(G(\mathbf{z}))-c)^2\right],$$

i.e. the discriminator learns to answer a for fakes and b for real data, while the generator wants to convince it to answer c on fakes. Usually one simply takes $a = 0$, $b = c = 1$.

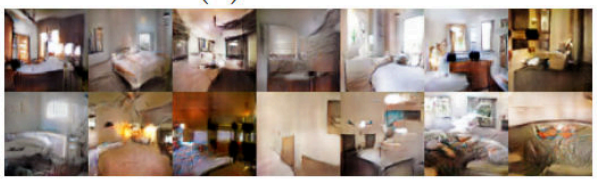


LSGAN

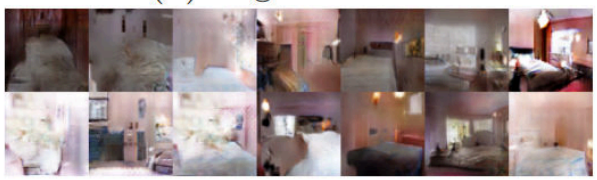
- Mao et al. showed that LSGAN is more robust to changes in architecture, suffers less from mode collapse; overall, the quadratic loss function is now often used.

(a) LSGANs.

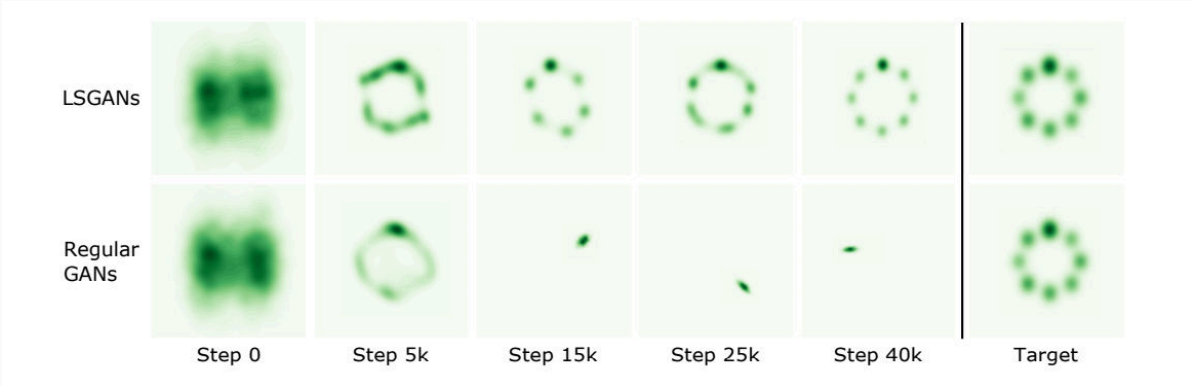
(b) Regular GANs.



(c) LSGANs.



(d) Regular GANs.



InfoGAN

- (Chen et al., 2016): InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets
- An important addition: it would be great to add disentanglement so that different features have meaning
- In ordinary GANs we just use a noise vector $\mathbf{z}$. In InfoGAN we split it into parts:
  - $\mathbf{z}$ is the truly incompressible noise, the source of entropy;
  - $\mathbf{c} = c_1 c_2 \dots c_L$ is the latent code.
- One can try adding this to an ordinary GAN so that the generator becomes $G(\mathbf{z}, \mathbf{c})$, but we need to add something so that $\mathbf{c}$ matters and G cannot simply ignore it.

InfoGAN

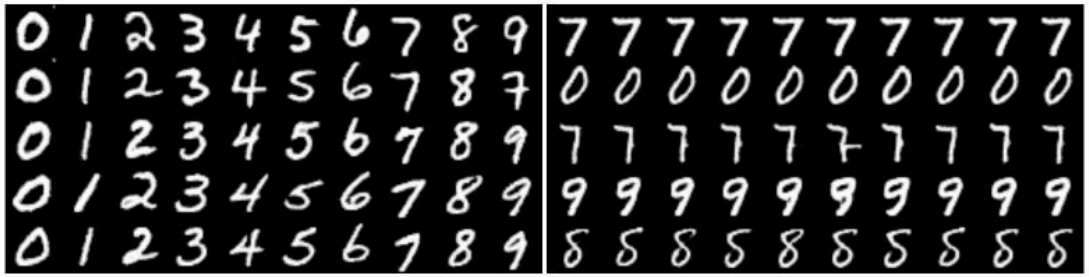
- InfoGAN: we will require that there be high mutual information $I(\mathbf{c}, G(\mathbf{z}, \mathbf{c}))$, i.e. the distribution $G(\mathbf{z}, \mathbf{c})$ should retain a lot of $\mathbf{c}$.
- Formally – a variational bound:

$$\begin{aligned} I(c; G(z,c)) &= H(c) - H(c|G(z,c)) \\ &= \mathbb{E}_{x\sim G(z,c)}[\mathbb{E}_{c'\sim P(c|x)}[\log P(c'|x)]] + H(c) \\ &= \mathbb{E}_{x\sim G(z,c)}[\underbrace{D_{\mathrm{KL}}(P(\cdot|x) \parallel Q(\cdot|x))}_{\geq 0} + \mathbb{E}_{c'\sim P(c|x)}[\log Q(c'|x)]] + H(c) \\ &\geq \mathbb{E}_{x\sim G(z,c)}[\mathbb{E}_{c'\sim P(c|x)}[\log Q(c'|x)]] + H(c) \end{aligned}$$

- And this lower bound we parametrize with a neural network Q via the reparametrization trick; it usually almost entirely coincides with D.

InfoGAN

- An example on MNIST with $\mathbf{c} = c_1 c_2 c_3$, where c_1 is a discrete label with 10 categories (without supervision! with a uniform prior distribution), and $c_2, c_3 \sim U[-1, 1]$:



(a) Varying c_1 on InfoGAN (Digit type)

(b) Varying c_1 on regular GAN (No clear meaning)

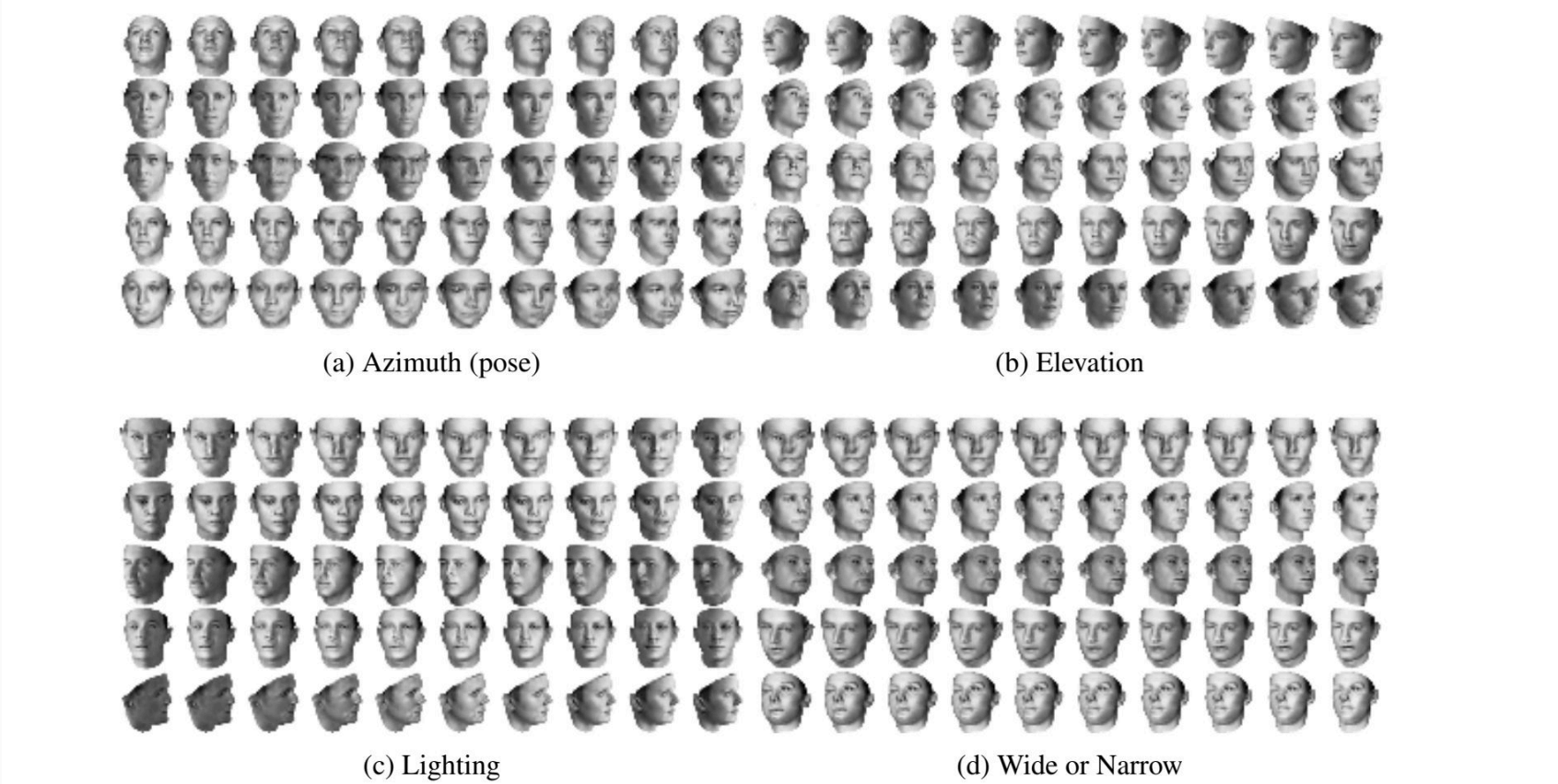
(c) Varying c_2 from -2 to 2 on InfoGAN (Rotation)

(d) Varying c_3 from -2 to 2 on InfoGAN (Width)

- In fact, using the code c_1 the classifier gives a 5% error on MNIST without any supervision.

InfoGAN

- 3D faces:



(a) Azimuth (pose)

(b) Elevation

(c) Lighting

(d) Wide or Narrow

InfoGAN

- 3D chairs:

(a) Rotation (b) Width

Wasserstein GAN

- Wasserstein GAN (Arjovsky et al., 2017):
  - what does it mean to learn a probability distribution?
  - it means minimizing $\mathrm{KL}(p_{\mathrm{data}} \| p_{\mathrm{model}})$;
  - but for this p_{model} has to exist, which is not the case if the distribution is concentrated on low-dimensional submanifolds; the manifold of p_{model} is unlikely to intersect substantially with the manifold of p_{data}, and KL will be undefined (infinite);
  - usually we add noise (Gaussian, for example), all models in ML have noise;
  - but for generative models noise gets in the way, making the generated samples blurry;
  - so usually noise is used for computing ML, but removed during generation, i.e. noise is wrong, but needed for ML.

Wasserstein GAN

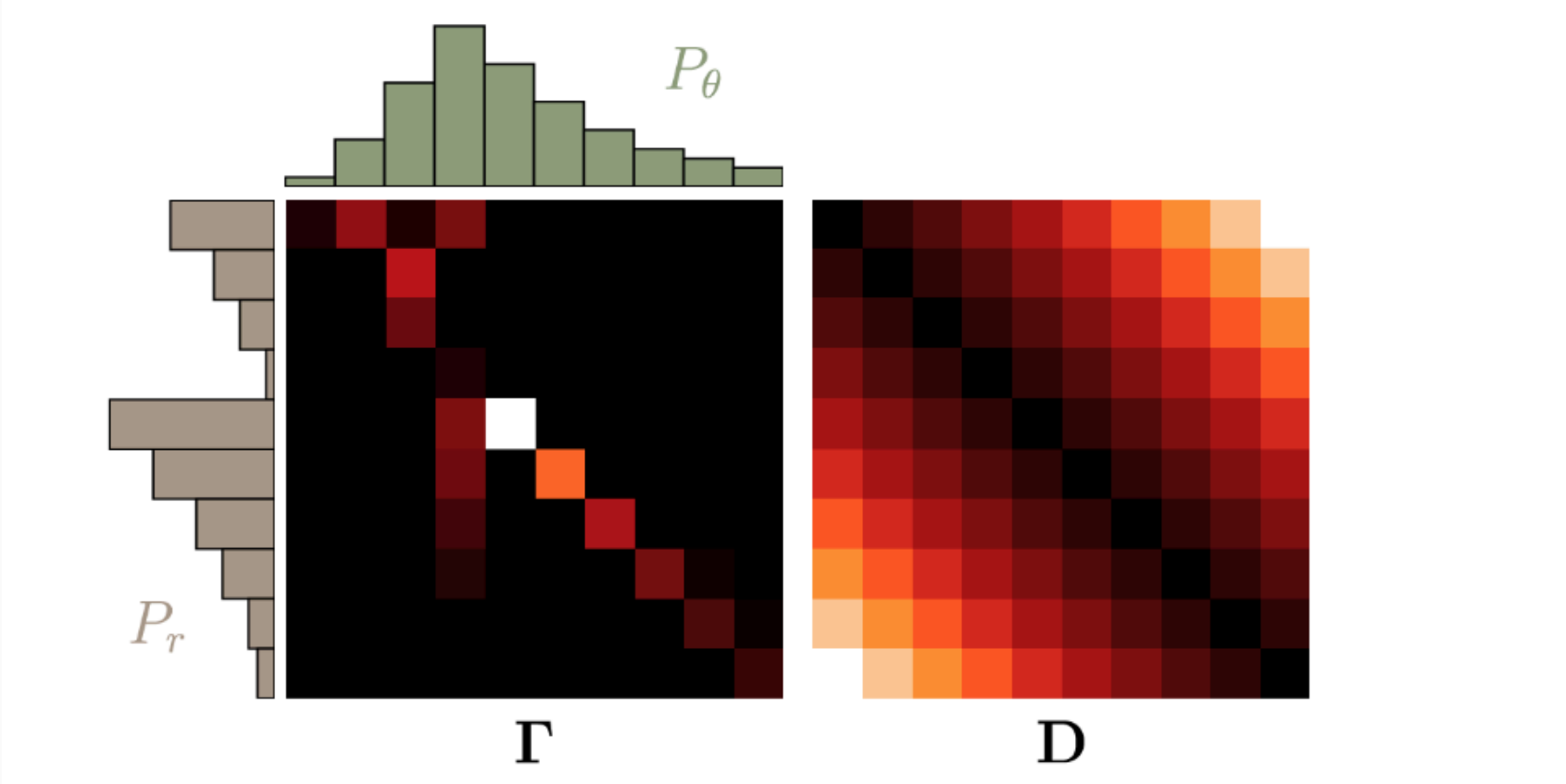
- Wasserstein GAN (Arjovsky et al., 2017): let's look at other metrics of closeness between p_{data} and p_{model}.
- Earth Mover distance, or Wasserstein-1:

$$W(p_{\text{data}}, p_{\text{model}}) = \inf_{\gamma \in \Pi(p_{\text{data}}, p_{\text{model}})} \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\gamma} \left[\|\mathbf{x} - \mathbf{y}\|\right],$$

  where $\Pi(p_{\text{data}}, p_{\text{model}})$ is the set of joint distributions $\gamma(\mathbf{x}, \mathbf{y})$ whose marginals are p_{data} and p_{model}.

Wasserstein GAN

- I.e. $\gamma(\mathbf{x}, \mathbf{y})$ shows how much "mass" needs to be redistributed from $\mathbf{x}$ to $\mathbf{y}$ in order to turn p_{data} into p_{model}.



Wasserstein GAN

- Example: consider $Z \sim U[0,1]$, $\mathbb{P}_0$ the distribution of $(0, Z)$ on $\mathbb{R}^2$, and $g_\theta(z) = (\theta, z)$, where θ is a parameter. Then

  - $W(\mathbb{P}_0, \mathbb{P}_\theta) = |\theta|$,
  - $JS(\mathbb{P}_0, \mathbb{P}_\theta) = \begin{cases} \log 2 & \text{if } \theta \neq 0\ , \\ 0 & \text{if } \theta = 0\ , \end{cases}$
  - $KL(\mathbb{P}_\theta \| \mathbb{P}_0) = KL(\mathbb{P}_0 \| \mathbb{P}_\theta) = \begin{cases} +\infty & \text{if } \theta \neq 0\ , \\ 0 & \text{if } \theta = 0\ , \end{cases}$
  - and $\delta(\mathbb{P}_0, \mathbb{P}_\theta) = \begin{cases} 1 & \text{if } \theta \neq 0\ , \\ 0 & \text{if } \theta = 0\ . \end{cases}$

- I.e. only the EM distance actually converges somewhere as $\theta \to 0$, and this is very much like learning a distribution concentrated on a submanifold.

Wasserstein GAN

- One can prove that $W(p_{\text{data}}, p_{\text{model}})$ is a continuous function of the parameters θ of the distribution p_{model} under fairly weak conditions.
- And the Kantorovich-Rubinstein duality says that the infimum

$$W(p_{\text{data}}, p_{\text{model}}) = \inf_{\gamma \in \Pi(p_{\text{data}}, p_{\text{model}})} \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\gamma} \left[\|\mathbf{x} - \mathbf{y}\|\right]$$

  is equivalent to

$$W(p_{\text{data}}, p_{\text{model}}) = \sup_{\|f\|_L \leq 1} \left(\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} \left[f(\mathbf{x})\right] - \mathbb{E}_{\mathbf{x}\sim p_{\text{model}}} \left[f(\mathbf{x})\right]\right),$$

  where the supremum is taken over all functions with Lipschitz constant ≤ 1.
- And we want to train a generative model $p_{\text{model}} = g_\theta(\mathbf{z})$.
- Let's parametrize everything with neural networks.

Wasserstein GAN

- A network $f_{\mathbf{w}}$ for the function f and a network for g_θ. Then:
  - for a given g_θ we train the weights $f_{\mathbf{w}}$, maximizing

$$\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\left[f(\mathbf{x})\right] - \mathbb{E}_{\mathbf{x}\sim p_{\text{model}}}\left[f(\mathbf{x})\right];$$

  - for a given $f_{\mathbf{w}}$ we compute

$$\begin{aligned}\nabla_\theta W(p_{\text{data}}, p_{\text{model}}) = \nabla_\theta\left(\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\left[f(\mathbf{x})\right] - \mathbb{E}_{\mathbf{x}\sim p_{\text{model}}}\left[f(\mathbf{x})\right]\right) = \\ = -\mathbb{E}_{\mathbf{z}\sim Z}\left[\nabla_\theta f_{\mathbf{w}}(g_\theta(\mathbf{z}))\right].\end{aligned}$$

- And to keep $f_{\mathbf{w}}$ Lipschitz, one can simply apply weight clipping to the gradients.

Wasserstein GAN

- So here is the training algorithm:

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

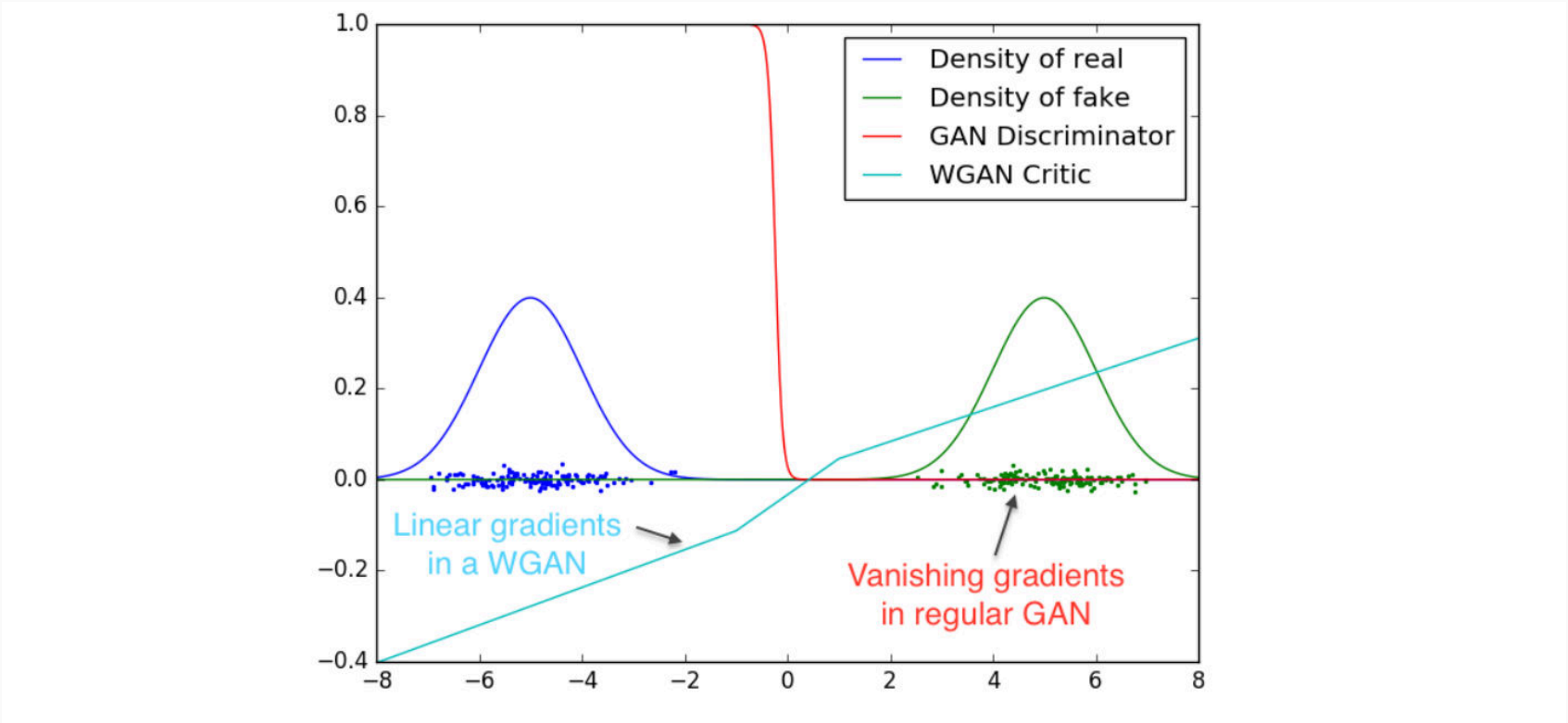
Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

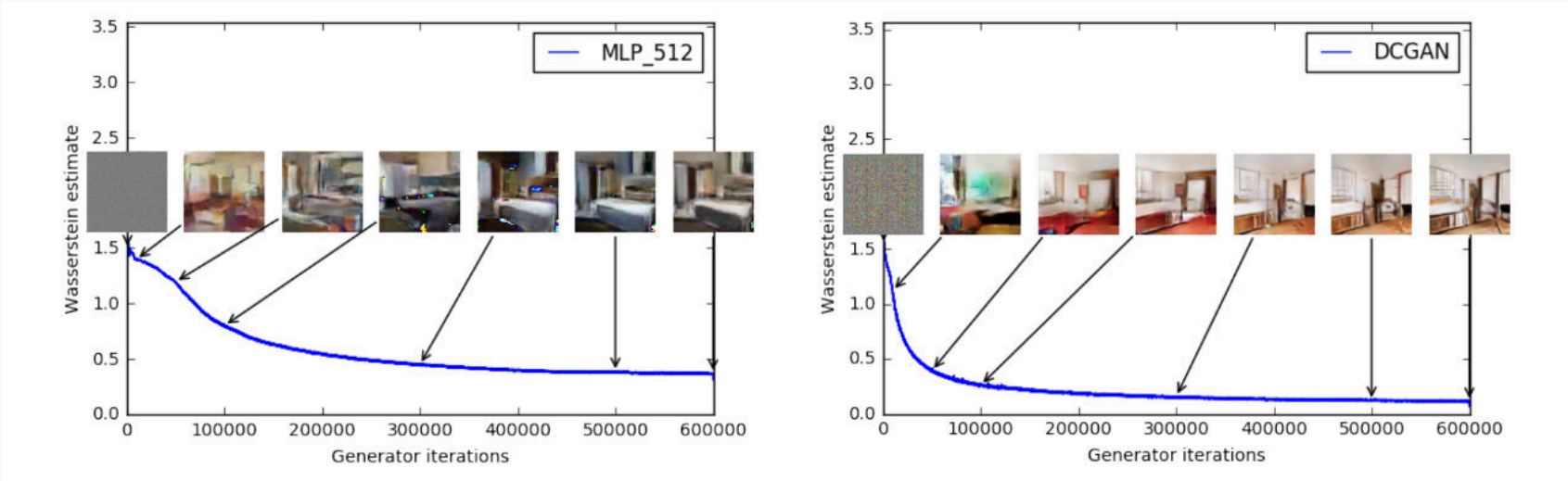
Wasserstein GAN

- An example showing that in WGAN one can safely train $f_{\mathbf{w}}$ to convergence, whereas in an ordinary GAN one perhaps should not train the discriminator that much:



Wasserstein GAN

- Quality is closely correlated with the loss function; on the left here is simply a fully connected network as the generator, on the right – DCGAN:

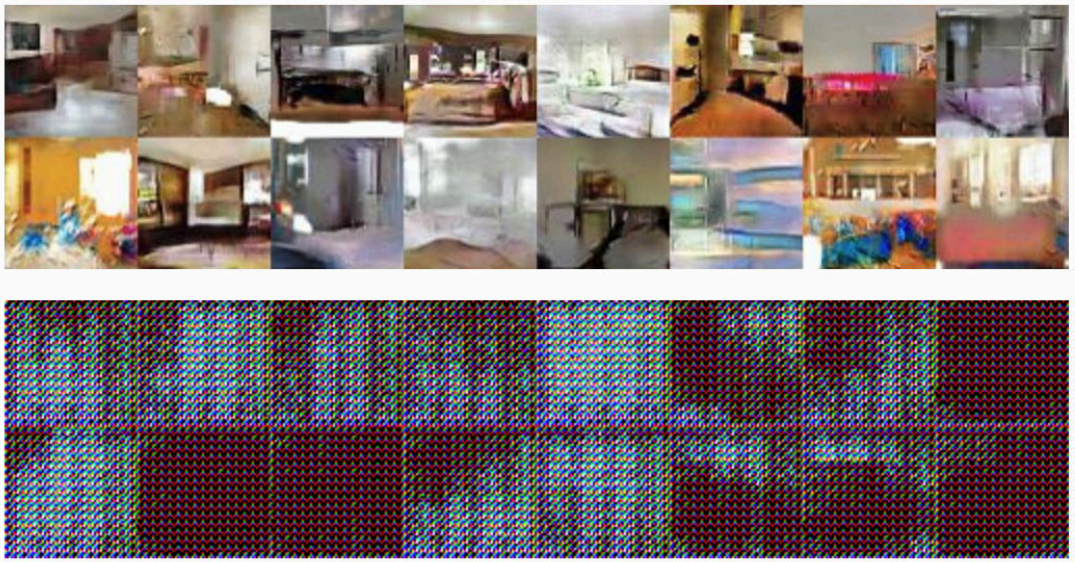


Wasserstein GAN

- WGAN is more robust to changes in architecture; here are ordinary WGAN and DCGAN (at the bottom):

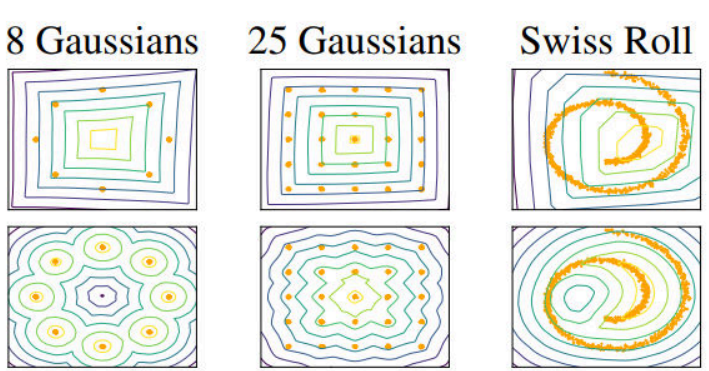
Wasserstein GAN

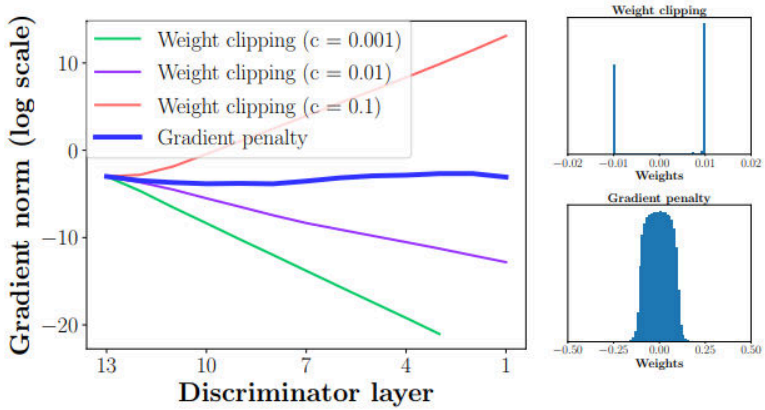
- And here is what happens if we remove batchnorm from the generator and use fewer filters (the same number at each level):



Wasserstein GAN

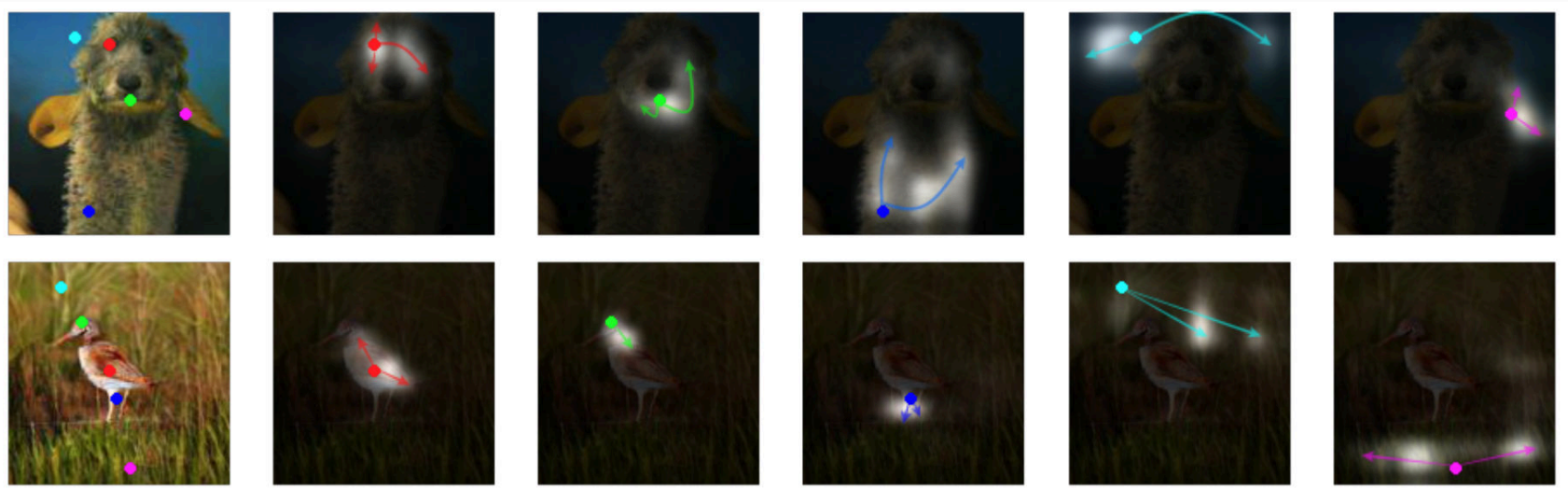
- (Gulrajani et al., 2017): Improved Training of Wasserstein GANs
- Weight clipping in the critic actually leads to problems, higher moments of the distributions are not learned and gradients explode/vanish
- It is better to use a soft regularizer on the gradient, like $\lambda \mathbb{E}_{\hat{\mathbf{x}} \sim P_{\hat{\mathbf{x}}}} \left[\left(\|\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})\|_2 - 1 \right)^2 \right]$





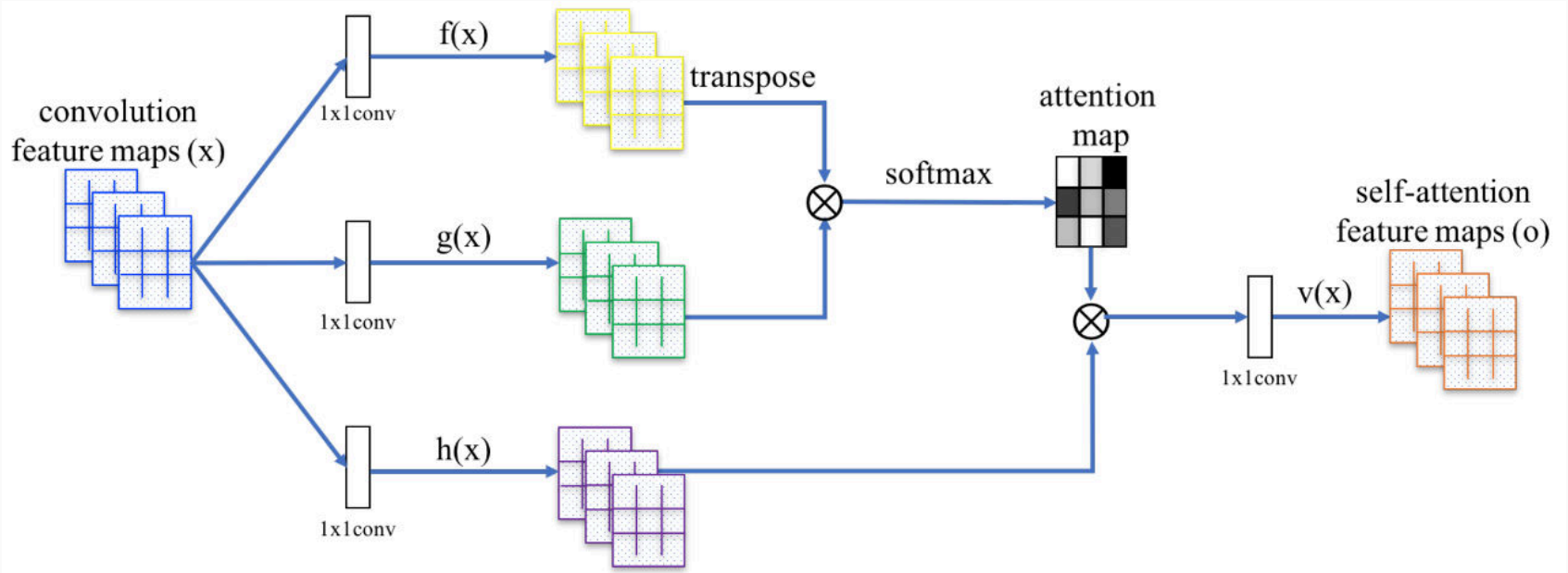
SAGAN

- (Zhang et al., 2019): Self-Attention Generative Adversarial Networks (SAGAN)
- They try to solve the problem of locality in generation by convolutional networks



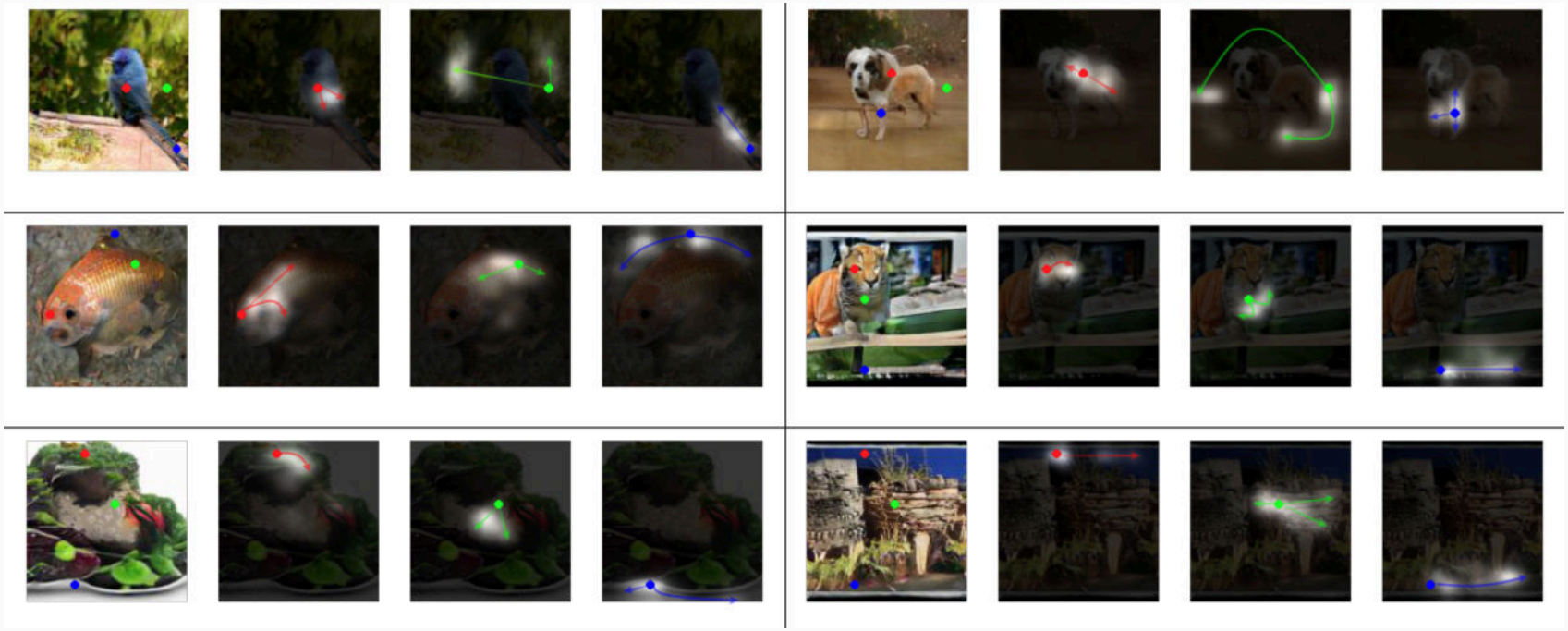
SAGAN

- Self-attention here is roughly as in the Transformer, only adapted to images:



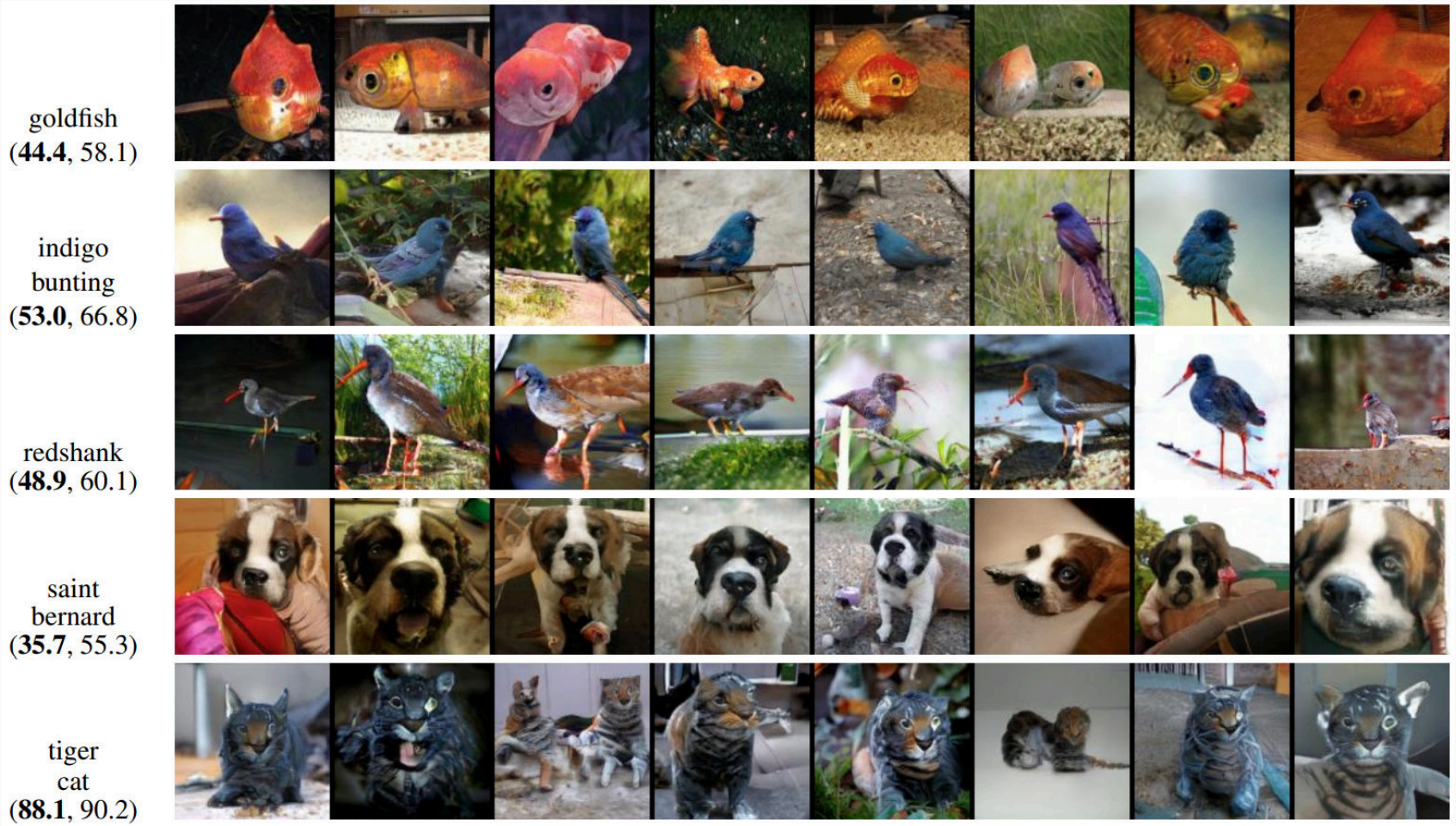
SAGAN

- And it turns out that the generator (here are examples from the last layer) can indeed look quite far away when it generates an image:



SAGAN

- In the original paper (Zhang et al., 2019) the images already come out good:



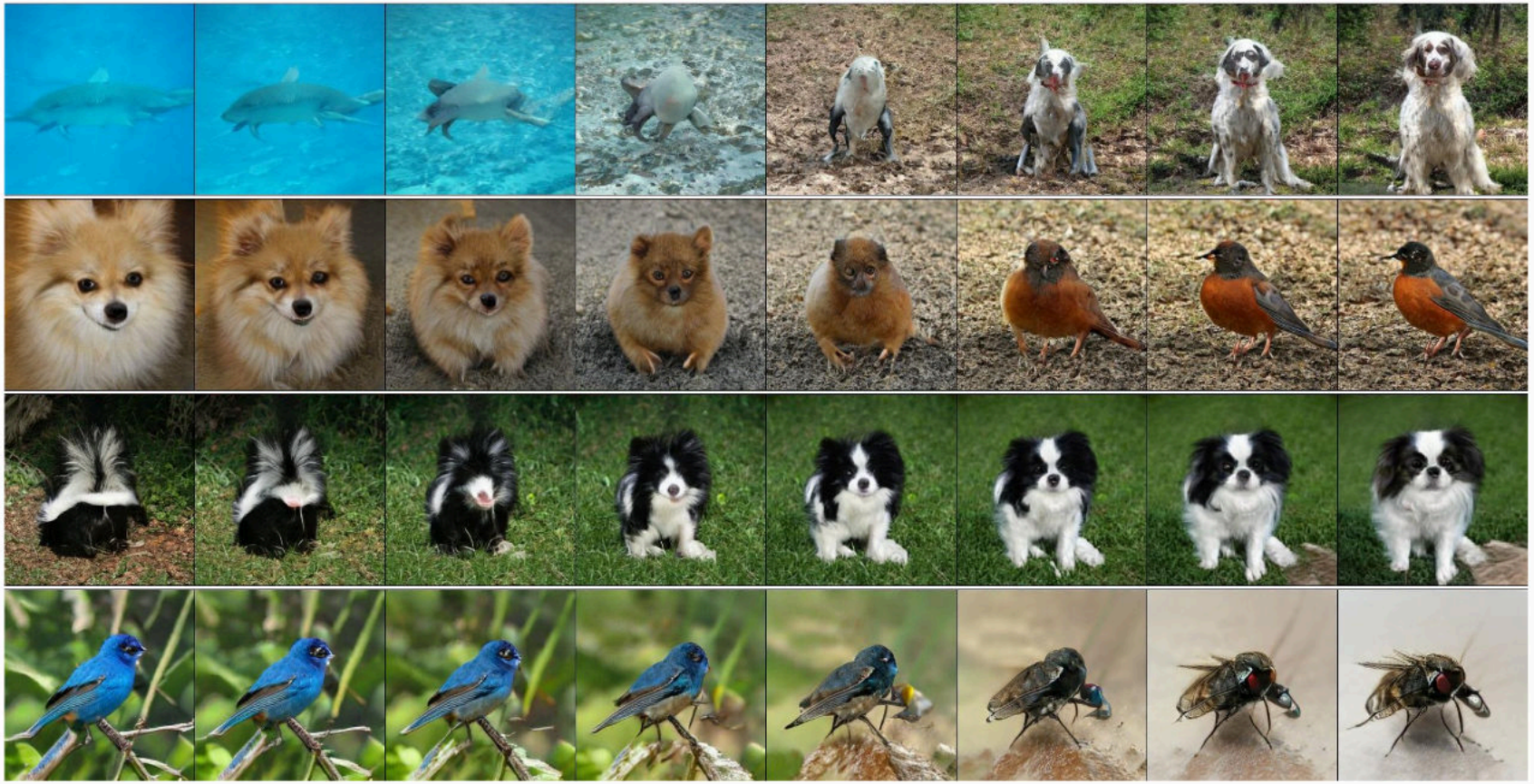
SAGAN

- The next stage – BigGAN (Brock et al., 2019), improved the Inception score threefold...

- They use SA-GAN (GAN with self-attention) with all the new tricks and special architectures

SAGAN

- Interpolations:



- Oh, but what does “better” mean?..

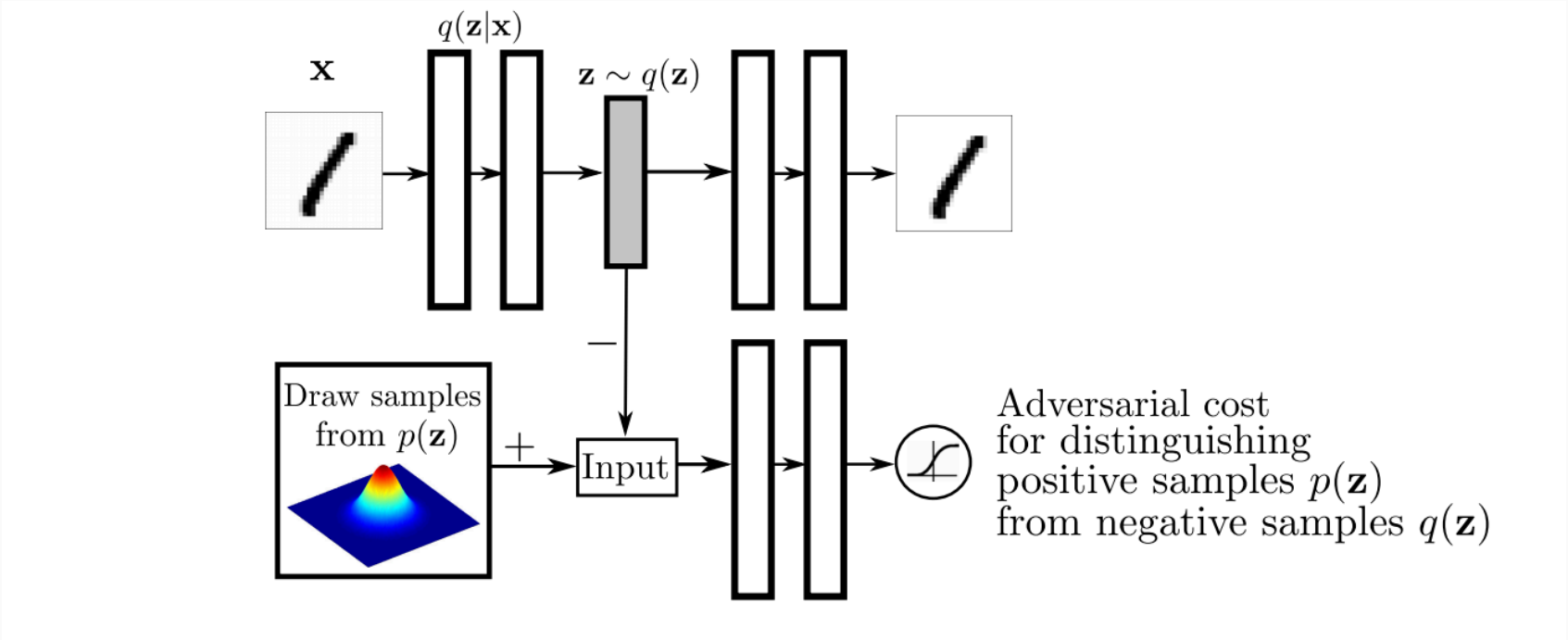
SAGAN

- It is hard to assess the quality of images.
- Inception score (Salimans et al., 2016; generally an important paper about GANs):
  - we take generated images $\mathbf{x}$;
  - we apply the standard Inception classifier;
  - we want $p(y \mid \mathbf{x})$ to have small entropy, but we want diversity, i.e. we want $p(y) = \int p(y \mid \mathbf{x} = G(\mathbf{z}))\mathrm{d}\mathbf{z}$ to have large entropy;
  - i.e. we get the metric $\exp(\mathbb{E}_{\mathbf{x}}\mathrm{KL}(p(y \mid \mathbf{x})\|p(y)))$;
  - it is hard to adapt this for training, but it works well for assessing quality.

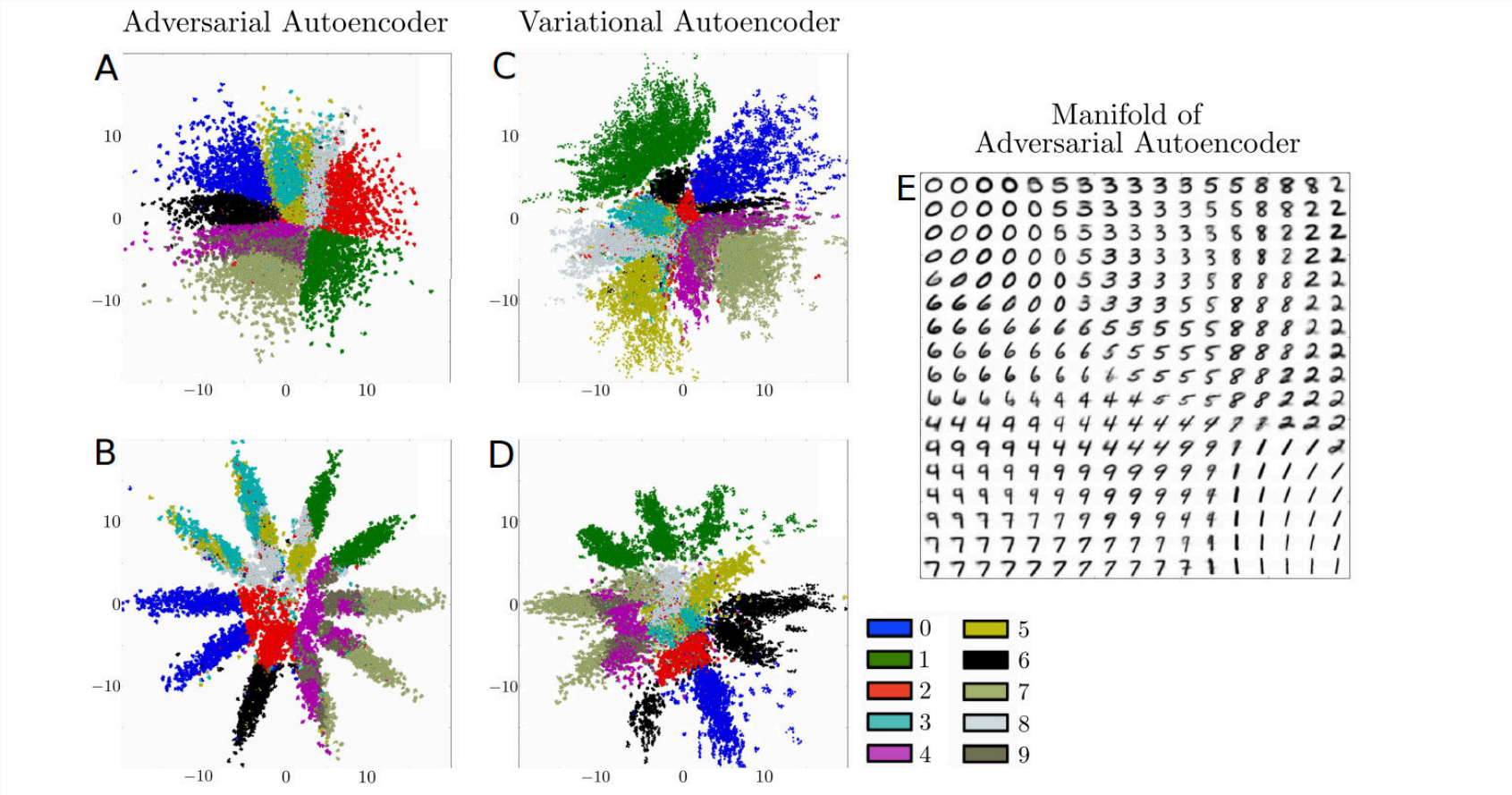
GAN-based architectures

AAE

- Adversarial autoencoders (Makhzani et al., 2015): the discriminator brings the distribution of features (in the hidden layer) to a given $p(\mathbf{z})$.

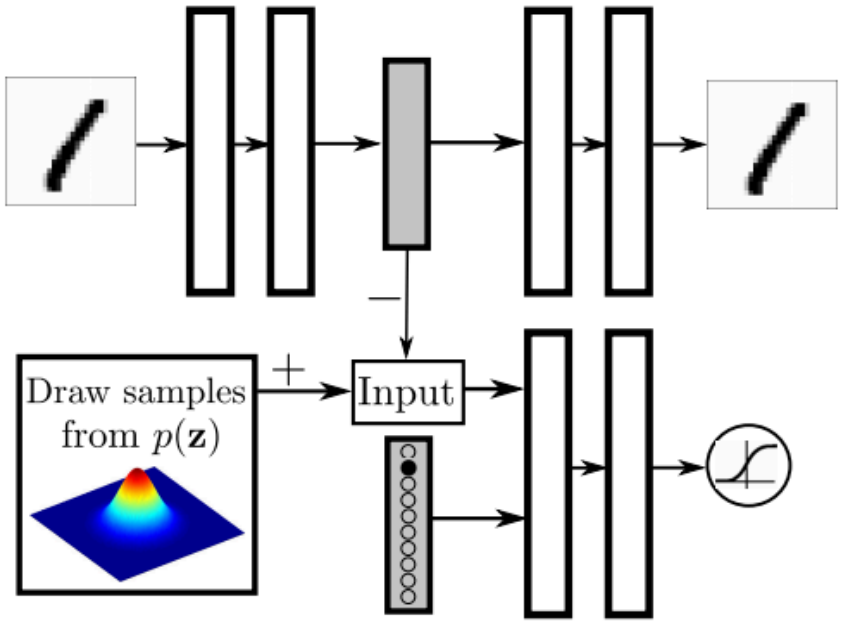


AAE

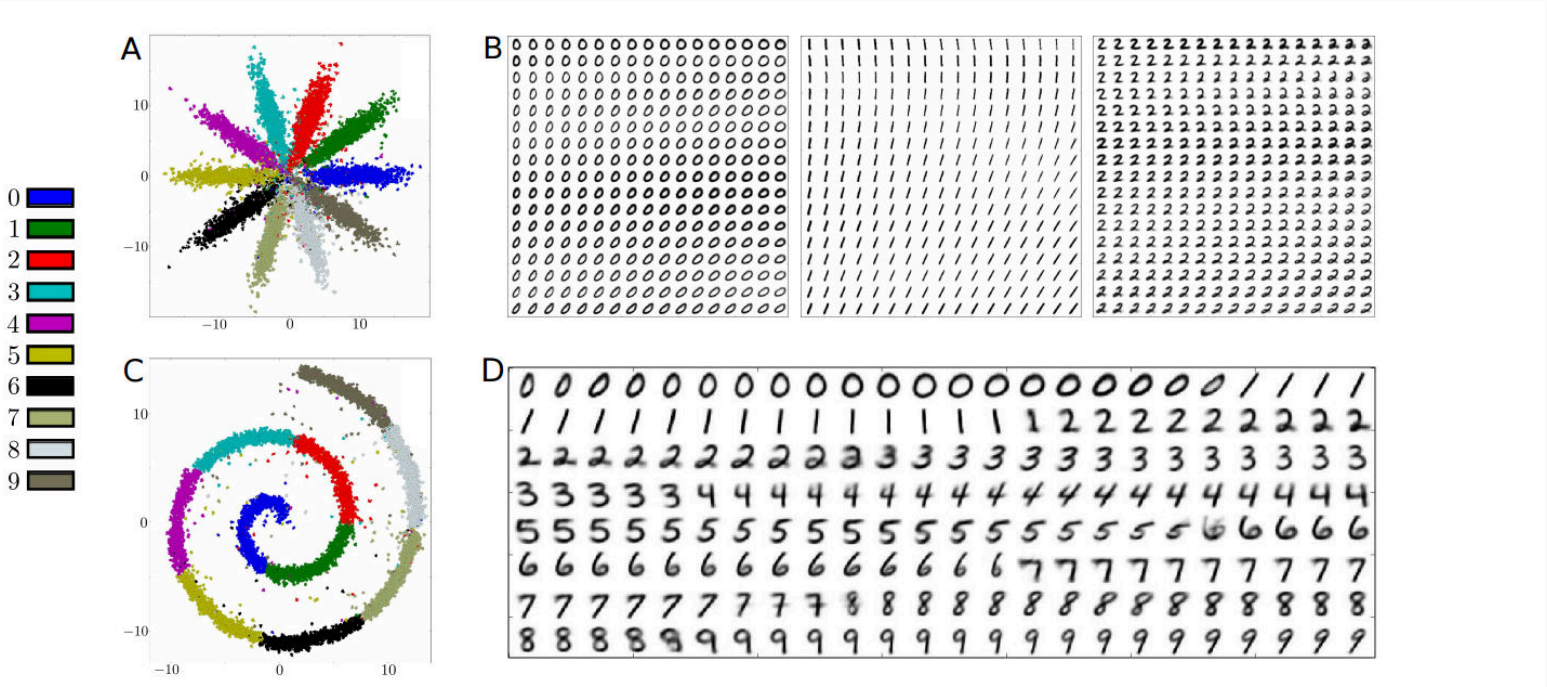

Adversarial Autoencoder
Variational Autoencoder
A
B
C
D
E
Manifold of
Adversarial Autoencoder
10
0
−10
0
1
2
3
4
5
6
7
8
9

AAE

- One can make the distributions conditional and obtain semi-supervised learning:

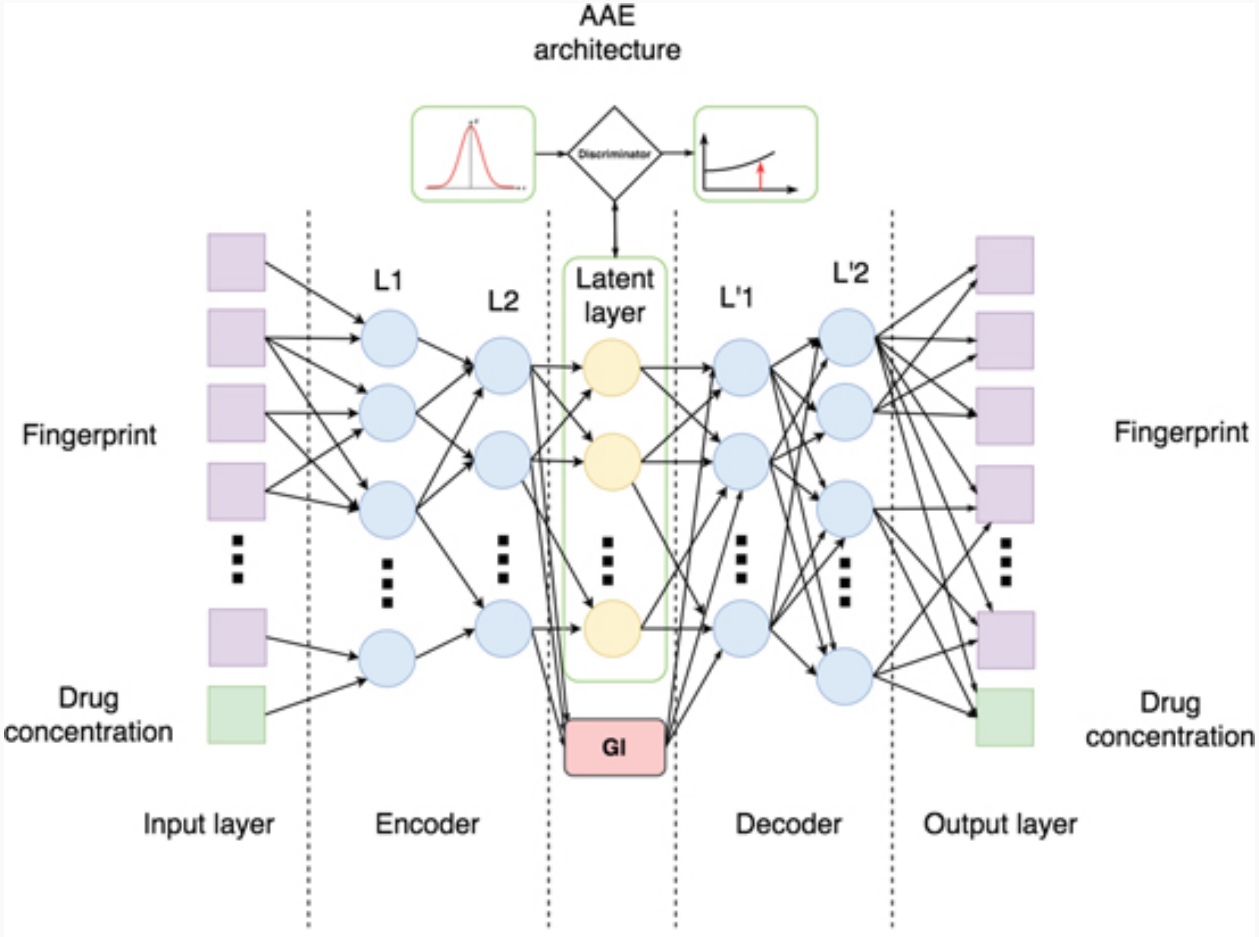


AAE



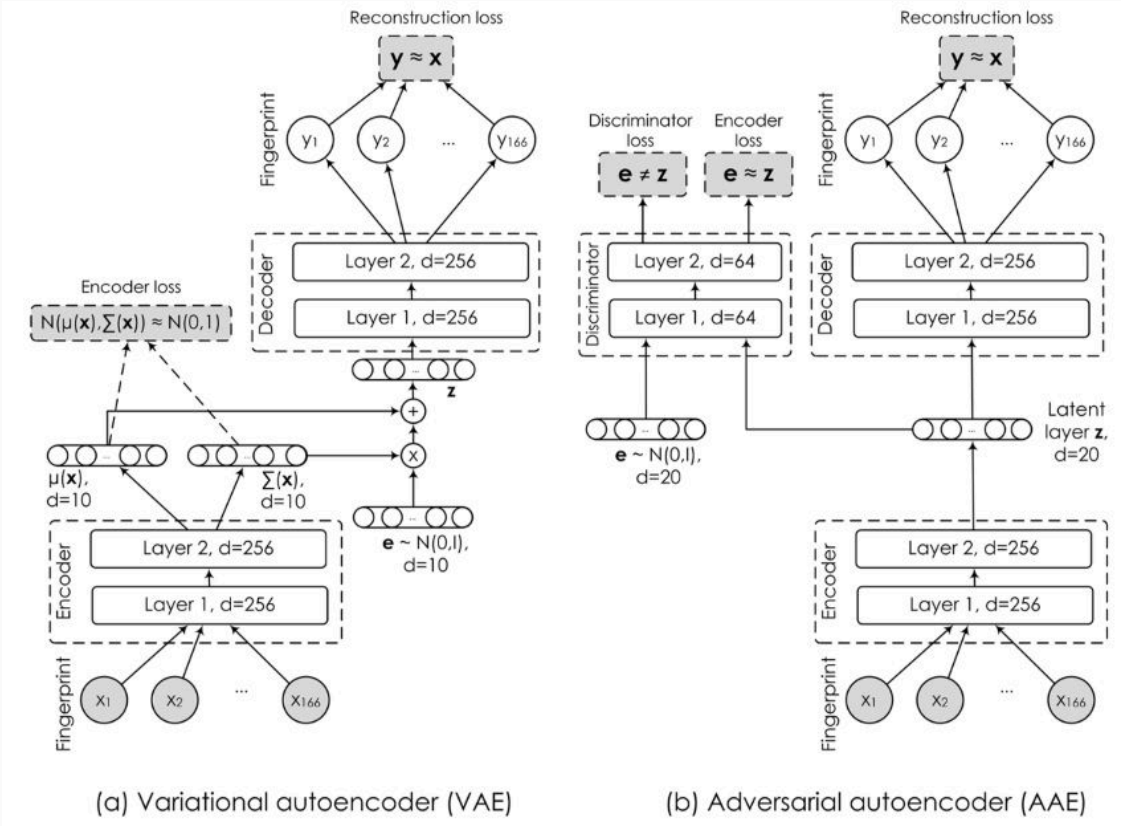
AAE

- (Kadurin et al., 2017) – AAE for generating molecules in oncology:



AAE

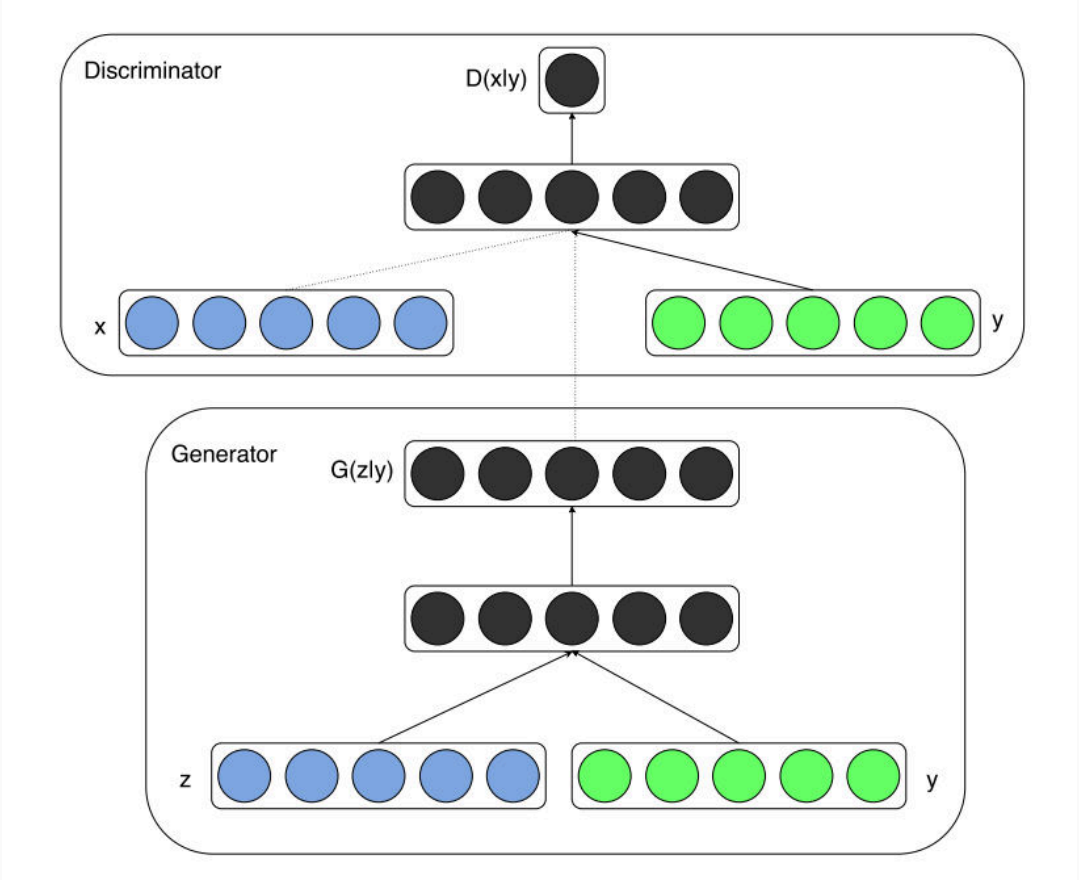
- (Kadurin et al., 2018) – druGAN for generating molecules; compared with VAE:



- (Polikovsky et al., 2018): MOSES – a platform for comparing generative models for molecules

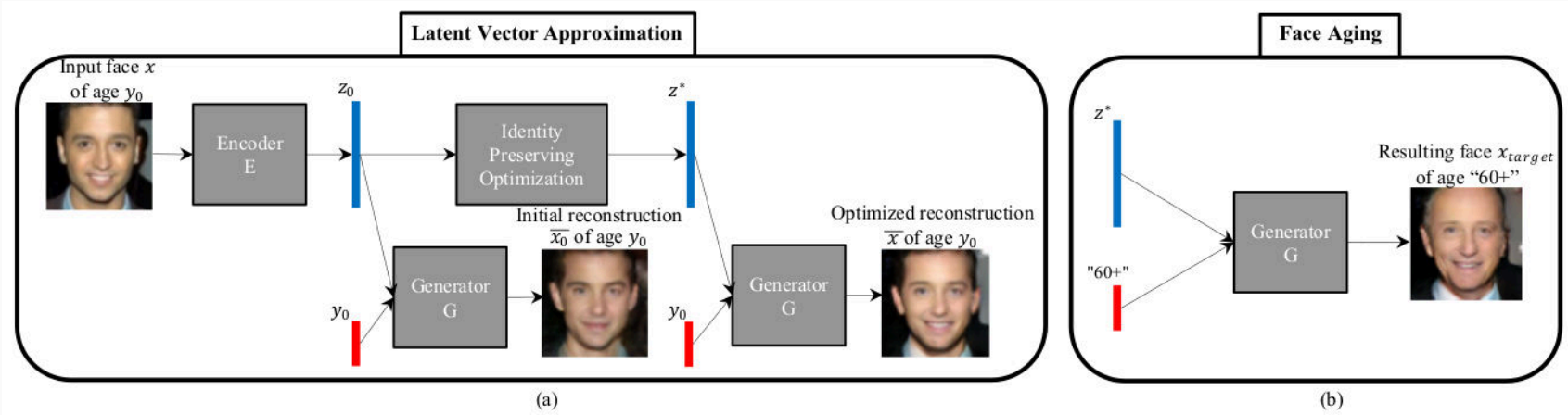
Conditional GANs

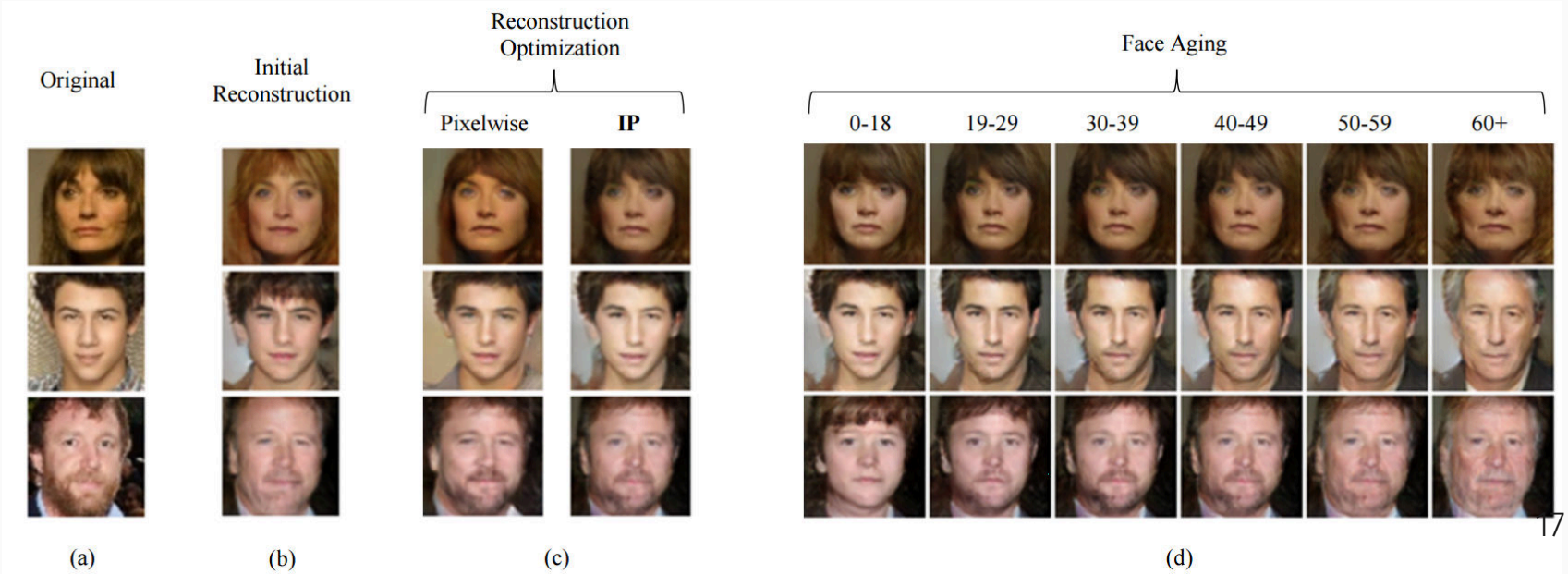
- Conditional GANs (Mirza and Osindero, 2014): G receives a random vector and an input vector, the input being the condition.



Conditional GANs

- (Antipov et al., 2017) – let's age a face with a conditional GAN:





Conditional GANs

- (Ledig et al., 2017) – ESRGAN for superresolution (but one could talk about this for a long time)

Stacked GANs

- It is often useful to chain several GANs in a row.
- (Huang et al., 2017): Stacked Generative Adversarial Networks
- G_i sequentially learns lower-level representations, inverting a deep network of E_i.
- New loss functions:
  - conditional loss – we train $\hat{\mathbf{h}}_i$ conditioned on $\mathbf{h}_{i+1}$, and so that the generator does not ignore the condition, we need $\mathbf{h}_{i+1}$ to be reconstructed through the corresponding encoder:

$$\mathcal{L}^{\text{cond}} = \mathbb{E}_{\mathbf{h}_{i+1}\sim p_{\text{data}}, \mathbf{z}_i \sim p_{\mathbf{z}_i}} \left[d(E_i(G_i(\mathbf{h}_{i+1}, \mathbf{z}_i)), \mathbf{h}_{i+1}) \right];$$

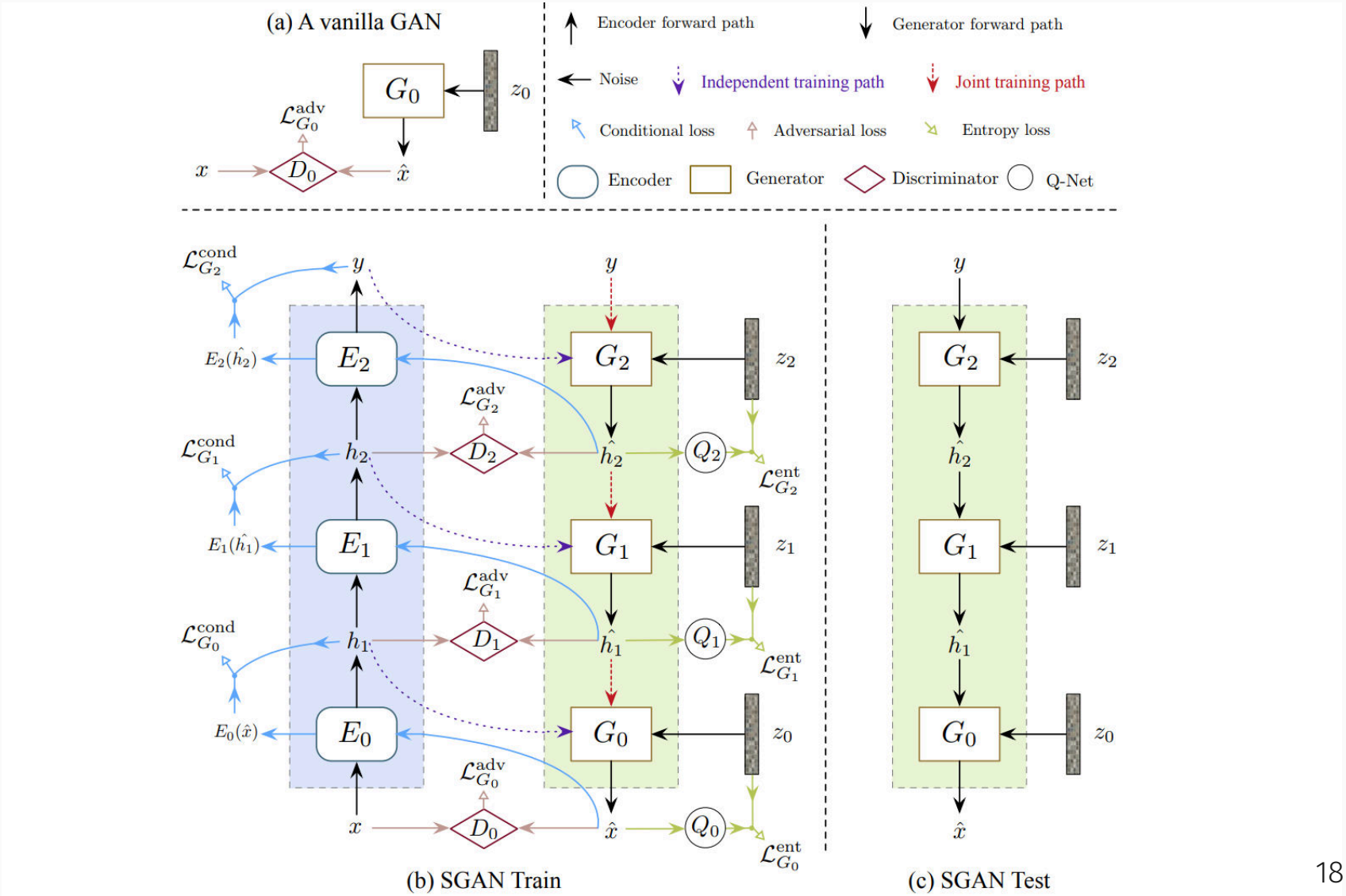
  - entropy loss – but we also need G_i not to ignore $\mathbf{z}$, we need to add diversity; for this we maximize entropy:

$$\mathcal{L}^{\text{ent}} = \mathbb{E}_{\mathbf{z}_i \sim p_{\mathbf{z}_i}} \left[\mathbb{E}_{\hat{\mathbf{h}}_i \sim G_i(\hat{\mathbf{h}}_i | \mathbf{z}_i)} \left[-\log Q_i(\mathbf{z}_i \mid \hat{\mathbf{h}}_i \right] \right],$$

    where Q_i is a neural-network-parametrized approximation for the posterior distribution $p_i(\mathbf{z}_i \mid \hat{\mathbf{h}}_i)$ (a variational bound).

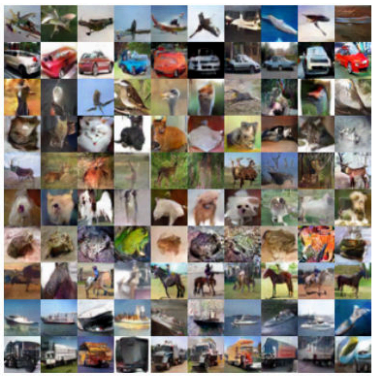
Stacked GANs

- SGAN:

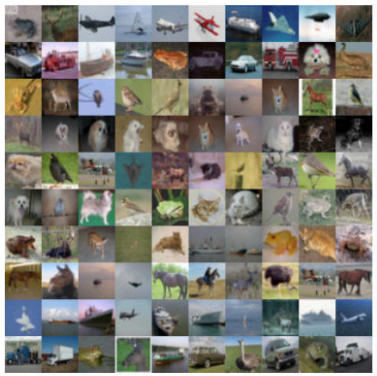


Stacked GANs

- It comes out better than its predecessors (though progressive growing is even better still):

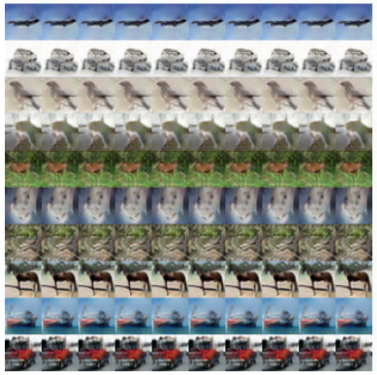


(a) SGAN samples (conditioned on labels)



(b) Real images (nearest neighbor)

(c) SGAN samples (conditioned on generated `fc3` features)



(d) SGAN samples (conditioned on generated `fc3` features, trained *without* entropy loss)

Method	Score
Infusion training [1]	4.62 ± 0.06
ALI [10] (as reported in [63])	5.34 ± 0.05
GMAN [11] (best variant)	6.00 ± 0.19
EGAN-Ent-VI [4]	7.07 ± 0.10
LR-GAN [65]	7.17 ± 0.07
Denoising feature matching [63]	7.72 ± 0.13
DCGAN† (with labels, as reported in [61])	6.58
SteinGAN† [61]	6.35
Improved GAN† [53] (best variant)	8.09 ± 0.07
AC-GAN† [43]	8.25 ± 0.07
DCGAN ($\mathcal{L}^{adv}$)	6.16 ± 0.07
DCGAN ($\mathcal{L}^{adv} + \mathcal{L}^{ent}$)	5.40 ± 0.16
DCGAN ($\mathcal{L}^{adv} + \mathcal{L}^{cond}$)†	5.40 ± 0.08
DCGAN ($\mathcal{L}^{adv} + L^{cond} + \mathcal{L}^{ent}$)†	7.16 ± 0.10
SGAN-no-joint†	**8.37** ± 0.08
SGAN†	**8.59** ± 0.12
Real data	11.24 ± 0.12

† Trained with labels.

Table 1: **Inception Score on CIFAR-10.** SGAN and SGAN-no-joint outperform previous state-of-the-art approaches.

Thank you!

Thank you for your attention!