ПОРОЖДАЮЩИЕ СОСТЯЗАТЕЛЬНЫЕ СЕТИ

Сергей Николенко

СПбГУ — Санкт-Петербург

30 октября 2025 г.

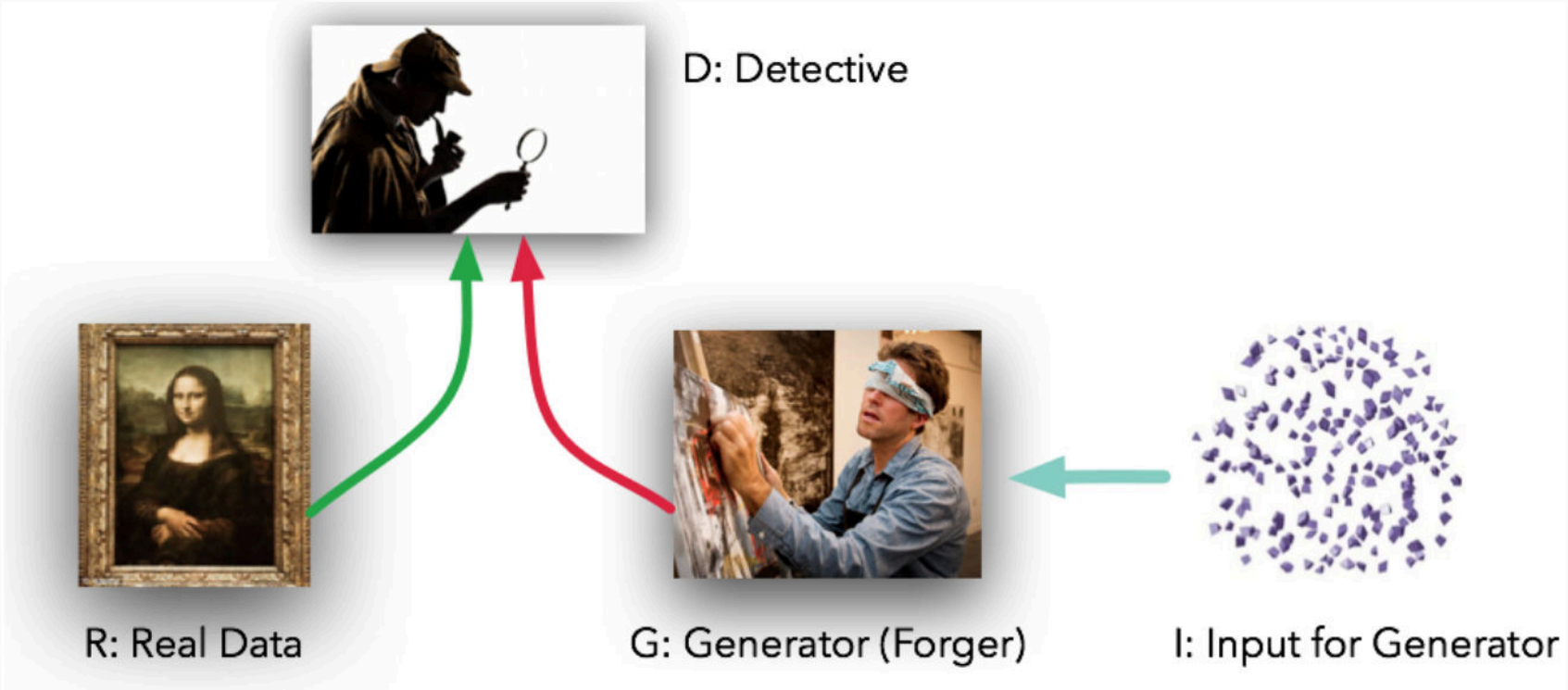
Random facts:

- 30 октября в России — День памяти жертв политических репрессий; 30 октября 1990 г. на Лубянской площади был установлен Соловецкий камень в память о них
- 30 октября 1905 г. Николай II ввёл Высочайший Манифест об усовершенствовании государственного порядка («Манифест 17 октября»)
- 30 октября 1907 г. русский физик Борис Розинг получил патент №18076 на «Способ электрической передачи изображений на расстояние» (то есть телевидение)
- 30 октября 1938 г. Орсон Уэллс поставил на радиостанции CBS «Войну миров», стилизованную под прямой репортаж с места событий; многие американцы поверили, и постановка вызвала настоящую панику; в ходе последовавших разбирательств выяснилось, что около 300 тысяч американцев лично видели марсиан
- 30 октября 1941 г. Франклин Рузвельт одобрил выделение 1 млрд долларов в качестве помощи Советскому Союзу
- 30 октября 1961 г. взорвалась самая мощная бомба в мировой истории: 58-мегатонная водородная бомба («Царь-бомба») на острове Новая Земля

Порождающие состязательные сети

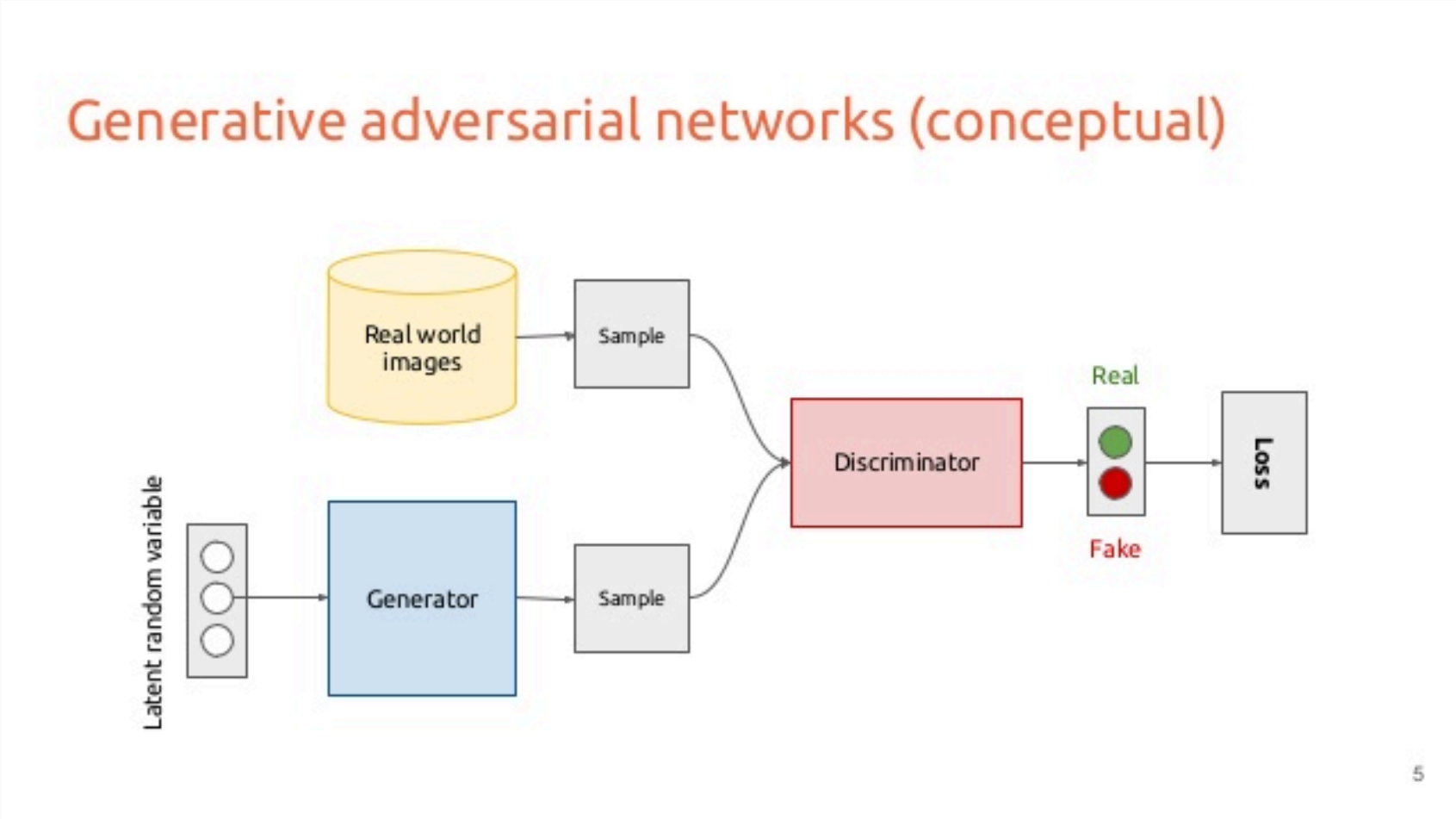
GAN

- Порождающие состязательные сети (Generative adversarial networks, GAN): генератор vs. дискриминатор, p_{data} vs. p_g.



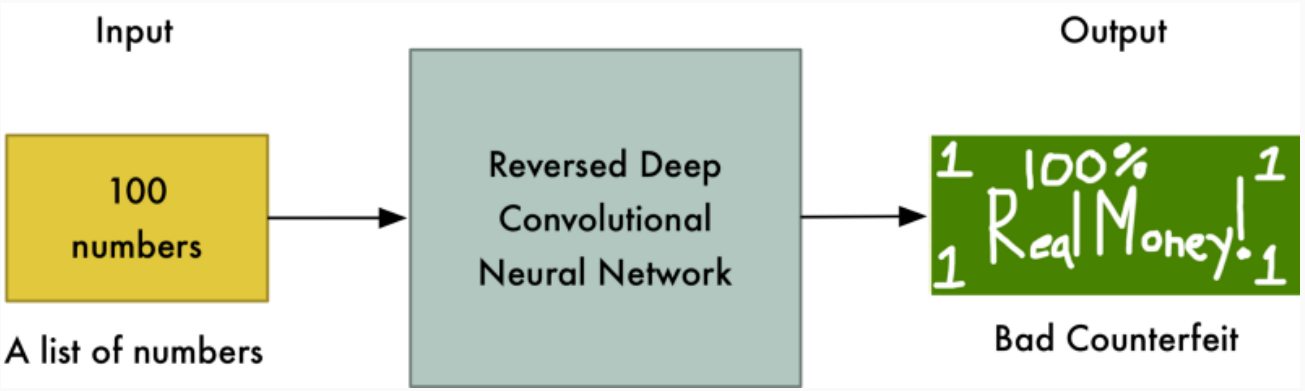
GAN

- Порождающие состязательные сети (Generative adversarial networks, GAN): генератор vs. дискриминатор, p_{data} vs. p_g.

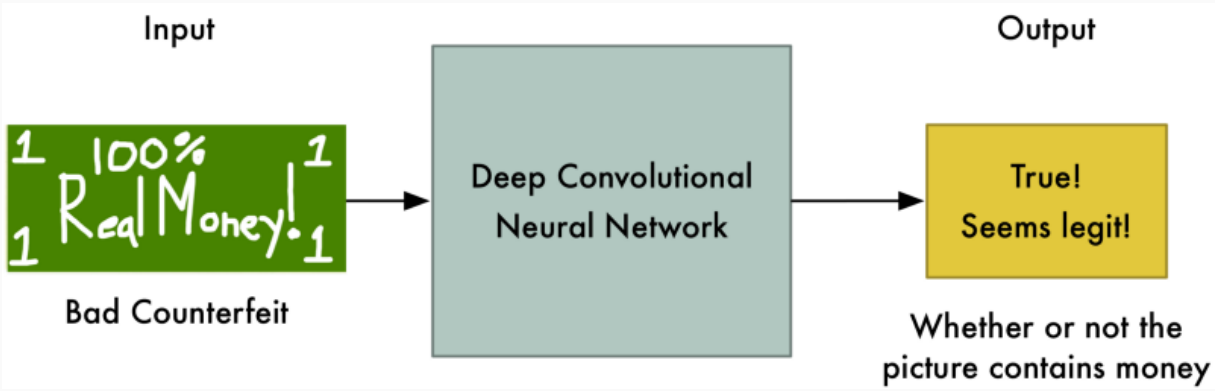


GAN

- В идеале должно быть примерно так (Geitgey, 2017):
  - сначала генератор плохой:

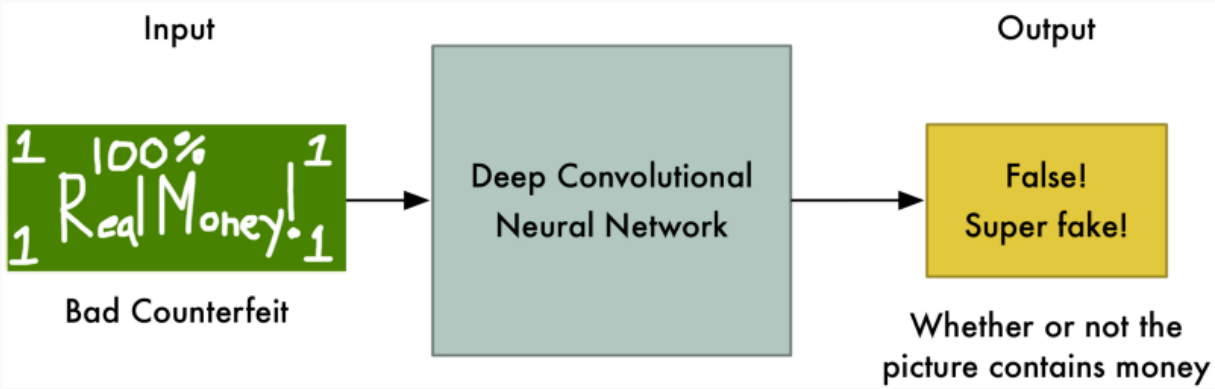


  - но и дискриминатор ничего не умеет:

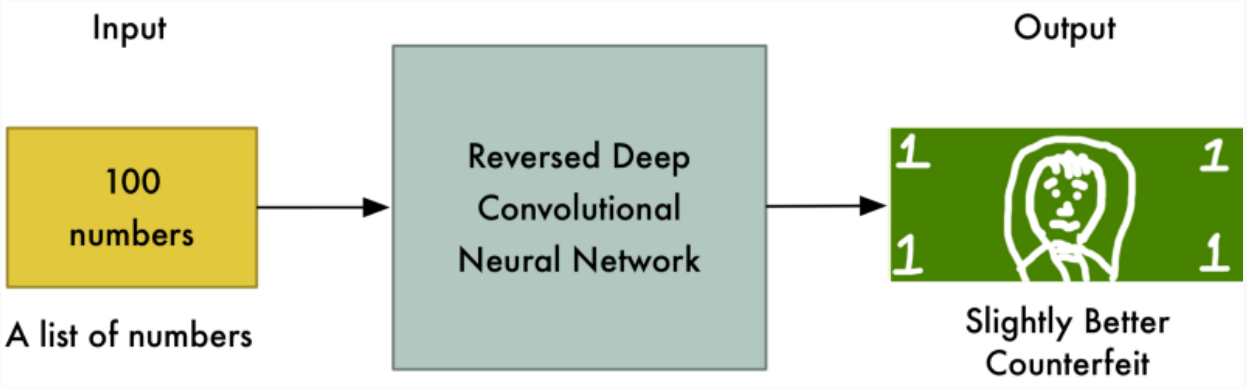


GAN

- В идеале должно быть примерно так (Geitgey, 2017):
  - поэтому сначала чуть обучим дискриминатор:



  - ...и генератору придётся придумать что-то получше:



- И так они будут продолжать, пока всё не получится.

GAN

- Формально, генератор – это функция

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

а дискриминатор – это функция

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

- Целевая функция для дискриминатора – это бинарная классификация:

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

где $p_g(\mathbf{x}) = G_{\mathbf{z} \sim p_z}(\mathbf{z})$ – распределение, порождённое G; то есть мы берём два мини-батча, с метками 0 из генератора и с метками 1 из данных.

GAN

- А генератор обучается обманывать дискриминатор, то есть минимизировать

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

- И теперь, если их объединить, получится типичная минимакс-игра:

$$\min_G \max_D V(D,G), \text{ где}$$

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

GAN

- Можно доказать, что $\max_D V(D, G)$ минимизируется именно тогда, когда $p_g = p_{\text{data}}$.
- Важное свойство: для фиксированного генератора G оптимальное распределение дискриминатора D — это распределение

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

- (Упражнение: попробуйте это доказать)

GAN

- Глобальный минимум критерия достигается тогда и только тогда, когда $p_g = p_{\text{data}}$; сам критерий для оптимального D эквивалентен

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

  это т.н. *дивергенция Йенсена–Шеннона* (Jensen–Shannon divergence).
- Но это всё результаты для ошибки генератора $\text{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$.
- А это не самая лучшая функция ошибки для генератора...

GAN

- Чем плохо минимизировать $\mathrm{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$?
  - перекрёстная энтропия хороша в классификаторе: если ответ неправильный, она не насыщается, а если правильный, насыщается и не меняется;
  - тут получается, что когда дискриминатор хорошо работает, его градиент обнуляется...
  - ...и градиент генератора тоже обнуляется!
- Поэтому на самом деле минимизируют

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

т.е. максимизируем вероятность того, что D ошибся, а не минимизируем вероятность его правильного ответа.

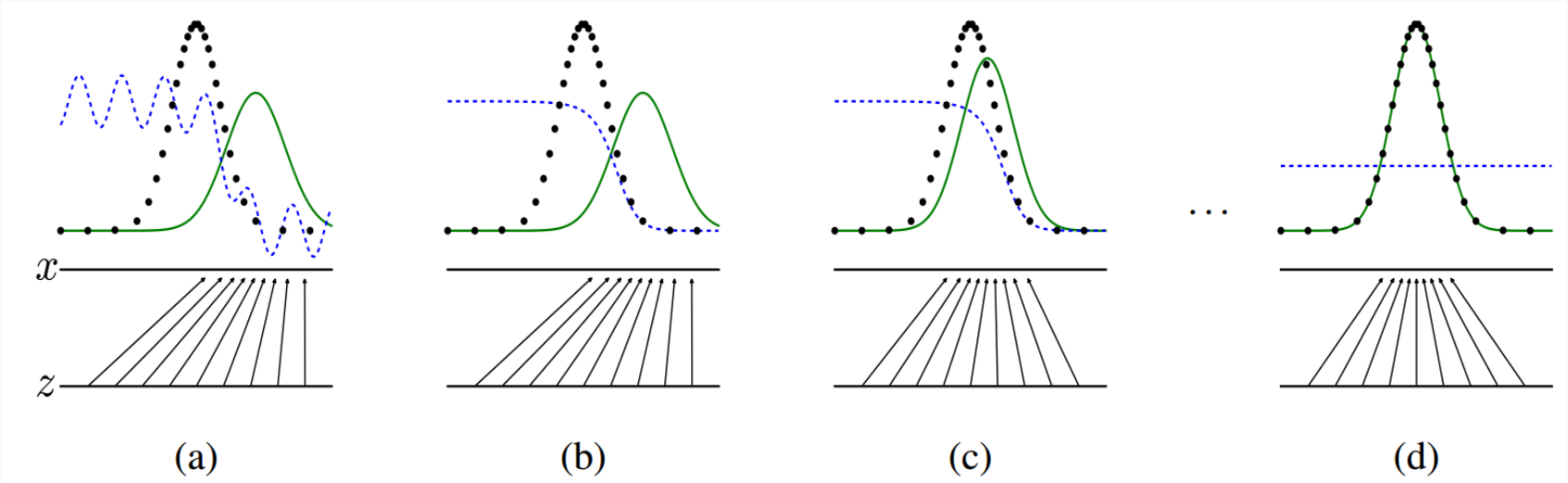
GAN

- Обучение похоже по сути на EM-алгоритм – попеременно:
  - фиксируем G, обновляем дискриминатор с функцией ошибки

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

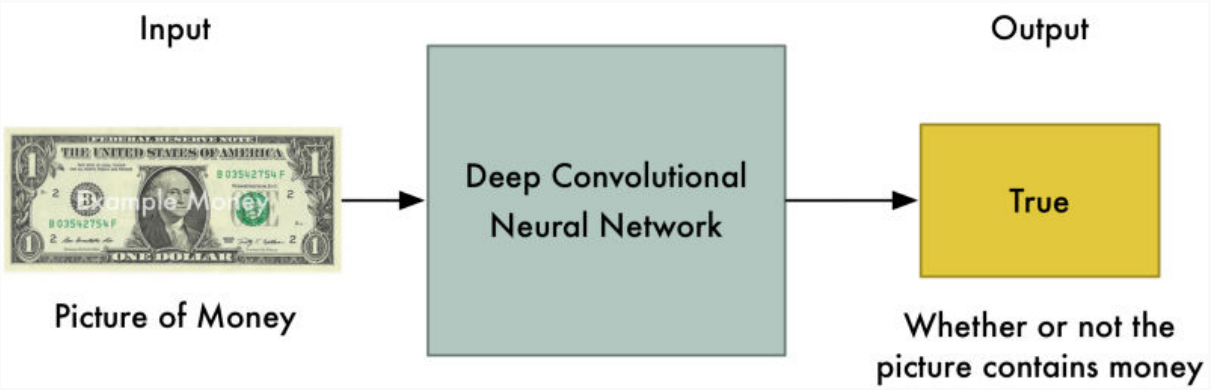
  - фиксируем D, обновляем генератор с функцией ошибки

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

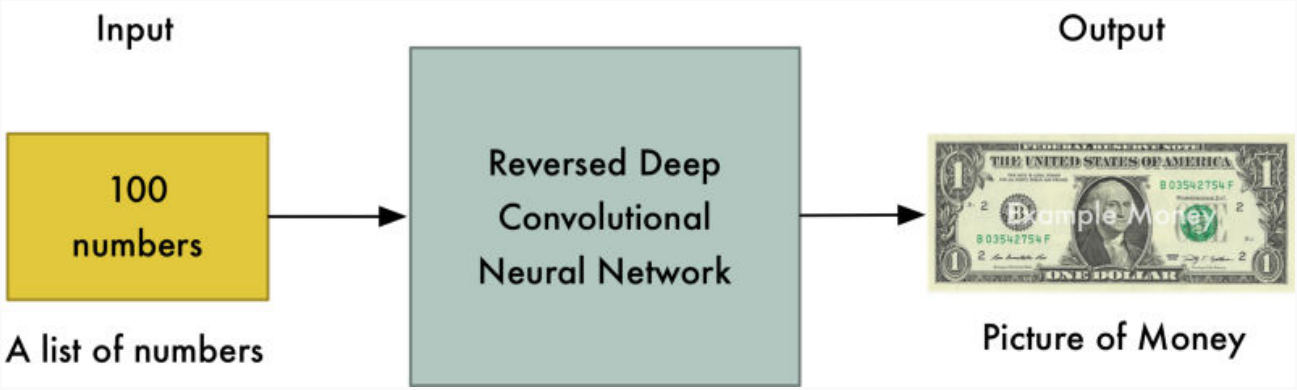


GAN

- В идеале должно быть примерно так (Geitgey, 2017):
  - дискриминатор пытается распознавать:

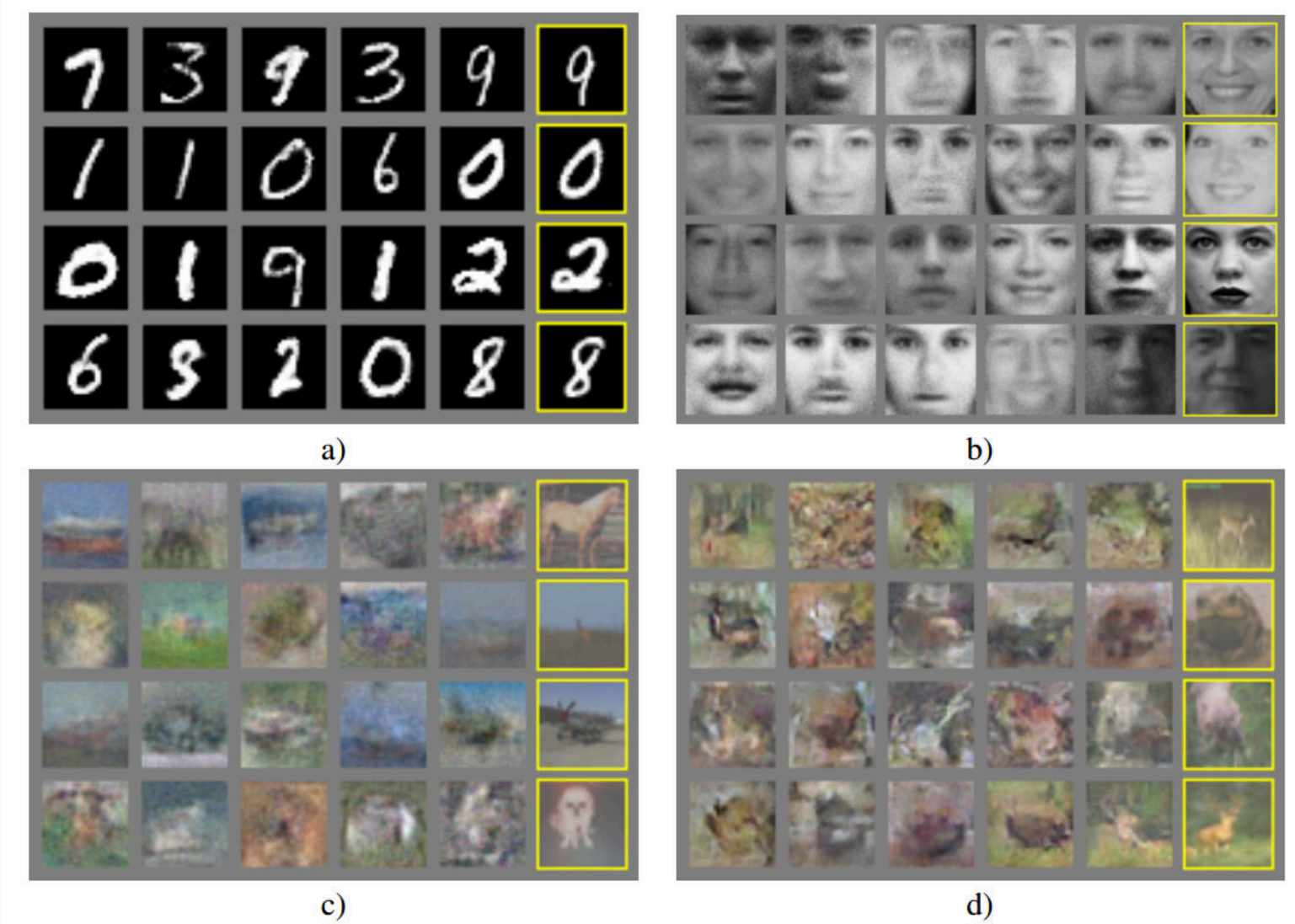


  - а генератор хочет порождать:



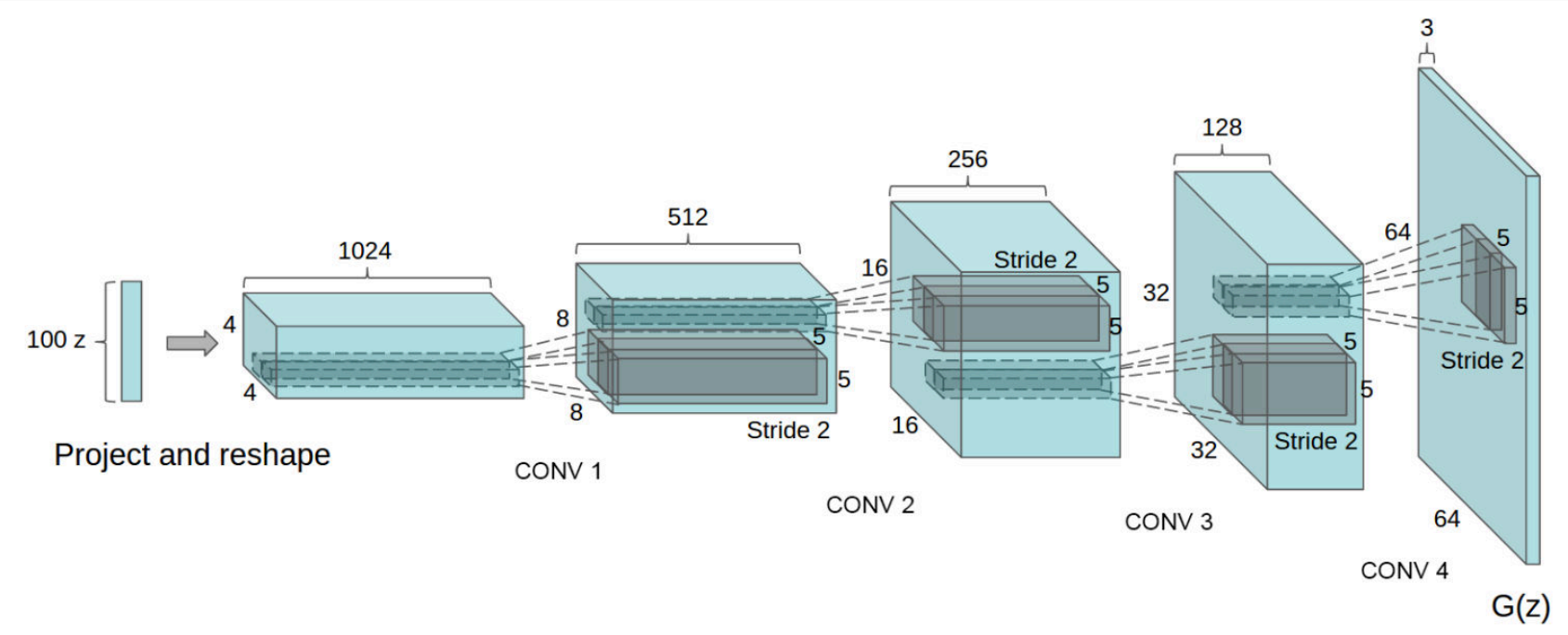
GAN

- Сэмплирование из GAN (Goodfellow et al., 2014):



DCGAN

- DCGAN – Deep Convolutional Generative Adversarial Networks (Radford et al., 2016).
- Несколько новых трюков (свёртки с пропусками вместо пулинга, везде batchnorm, убираем полносвязные слои, Adam...)



DCGAN

- Спальни (LSUN) после одной эпохи:

DCGAN

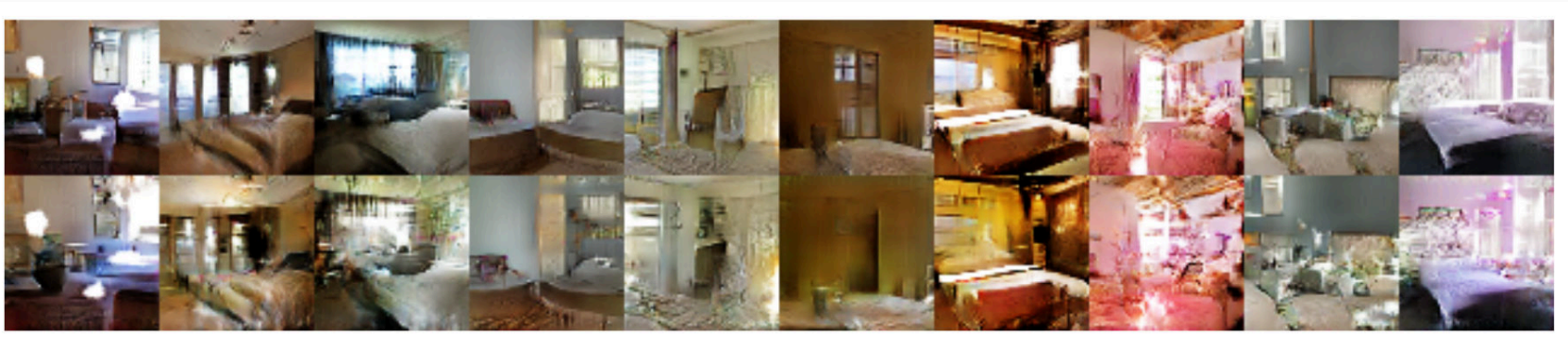
- Спальни (LSUN) после пяти эпох:

DCGAN

- Прогулка по пространству признаков:

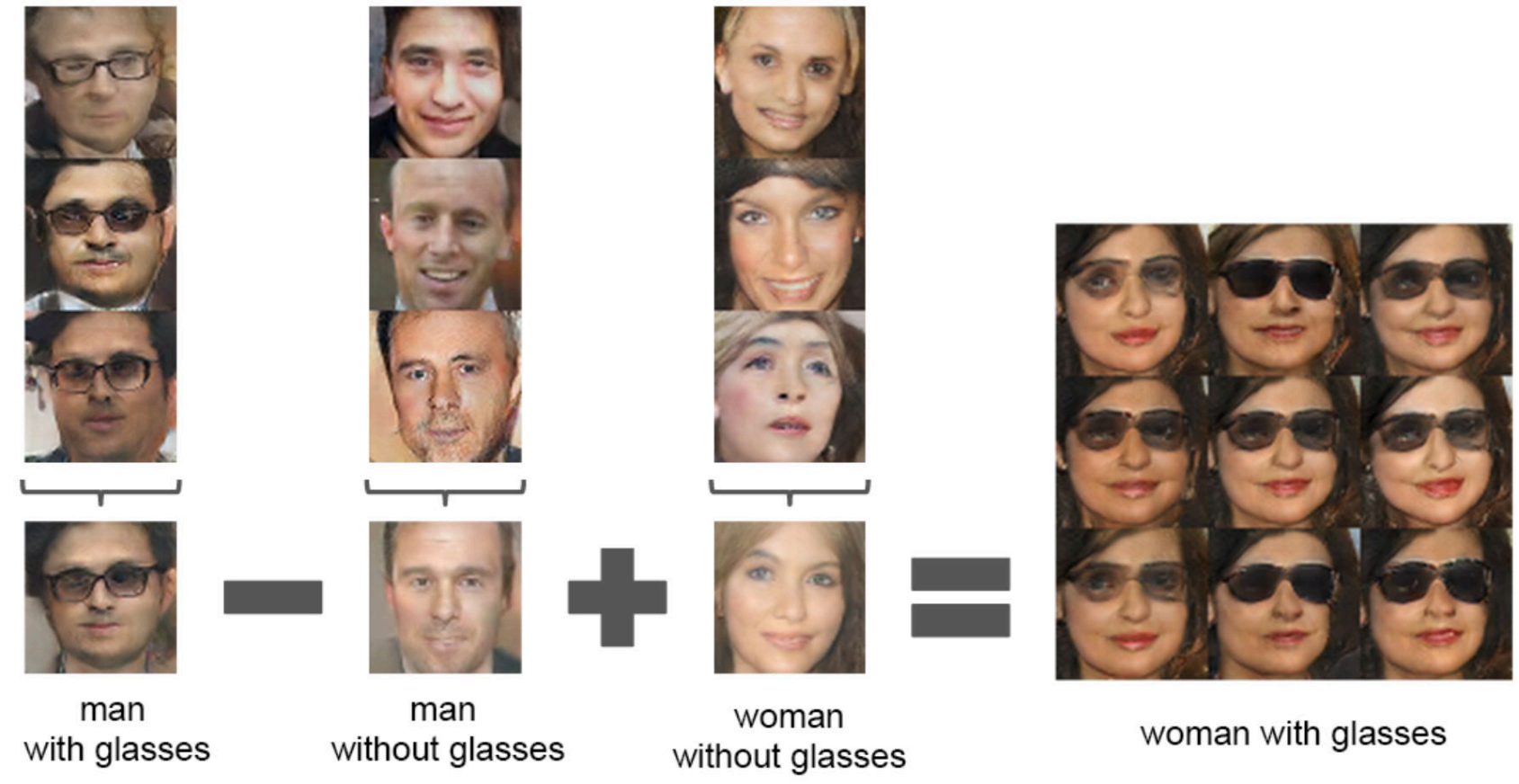
DCGAN

- Убираем фильтры, распознающие окна:



DCGAN

- Векторная арифметика (как в *word2vec*):



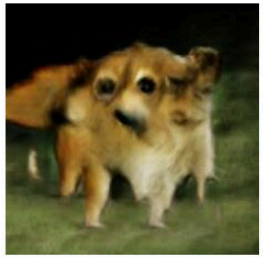
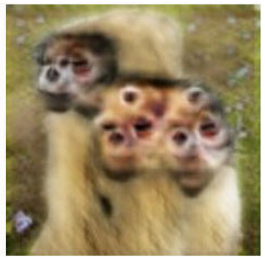
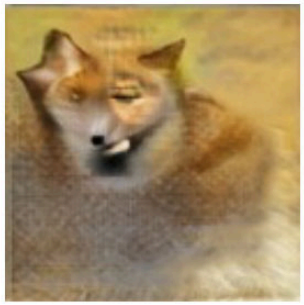
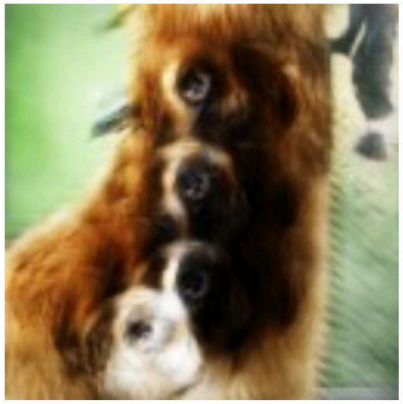
DCGAN

- (Geitgey, 2017): pixel art из игр NES

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

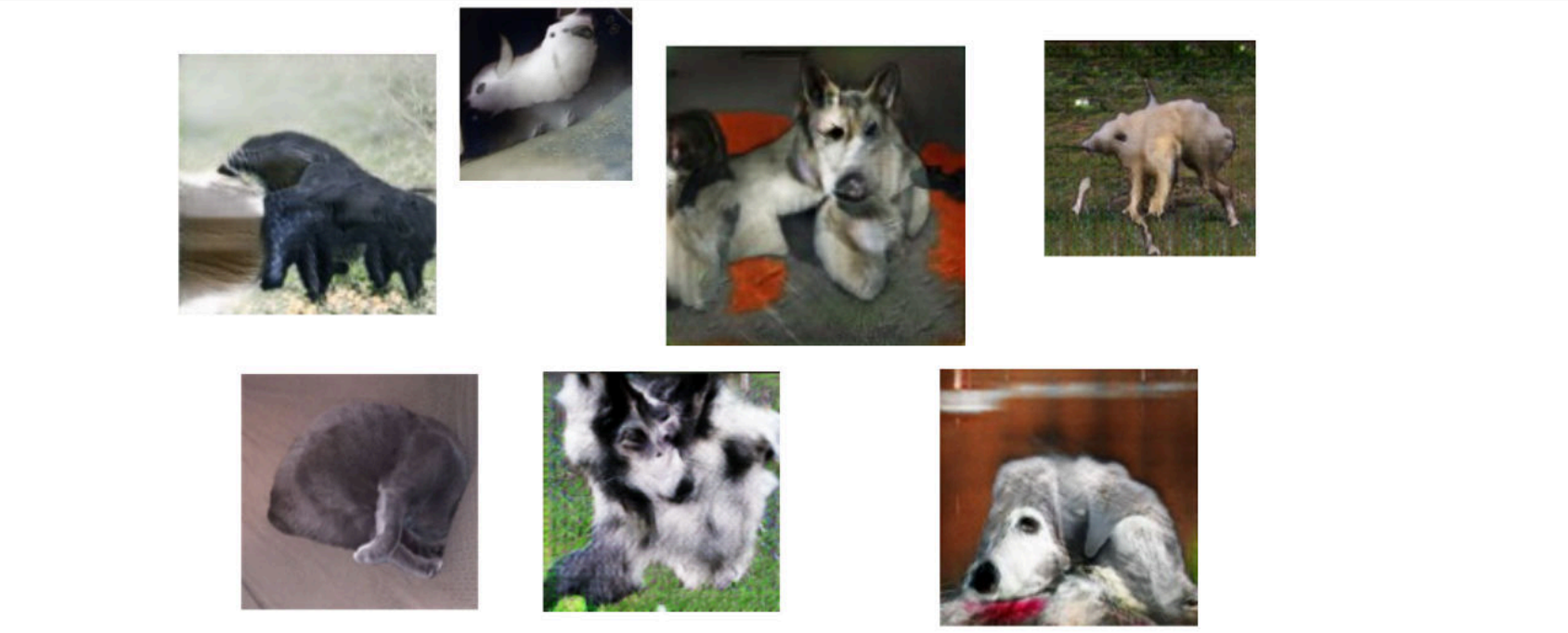
DCGAN

- Но не всегда, конечно, работает – проблемы с подсчётом:


DCGAN

- Но не всегда, конечно, работает – проблемы с перспективой (right?):



Прогресс

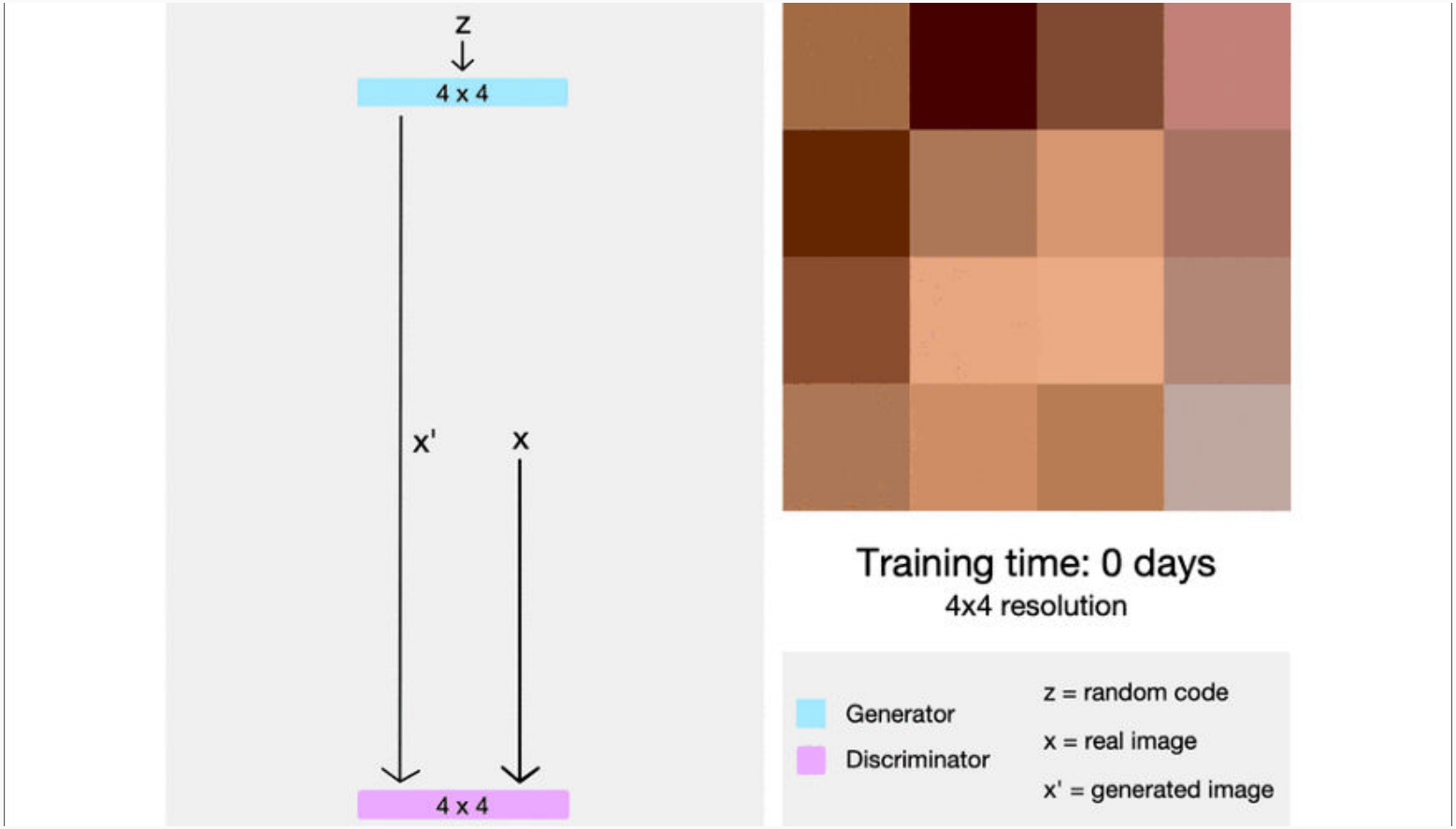
- Так работали самые первые GAN'ы.
- Но мы с тех пор прошли долгий путь:

- Как же так получилось?

Enlarge your GAN

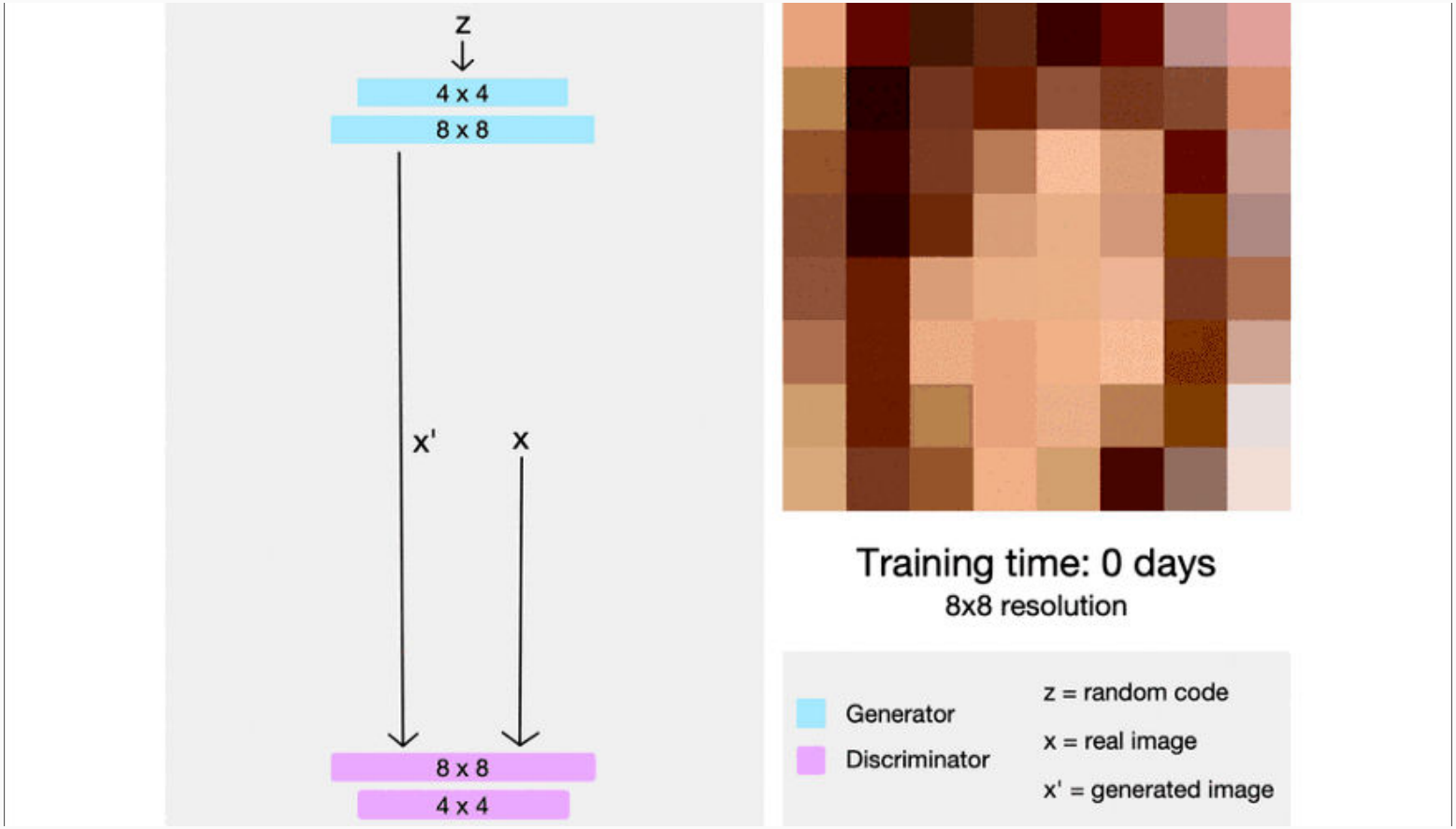
ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024



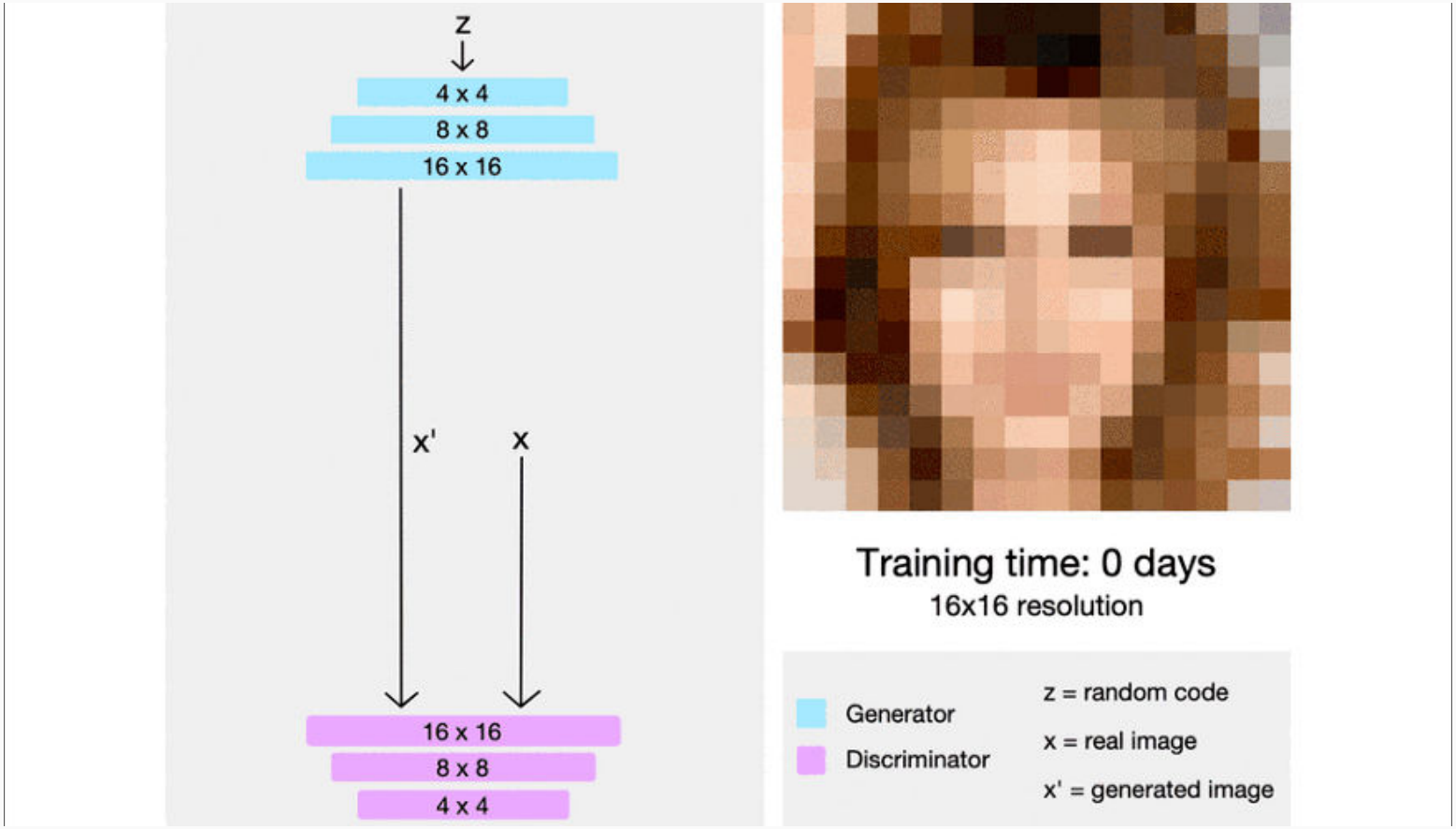
ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024



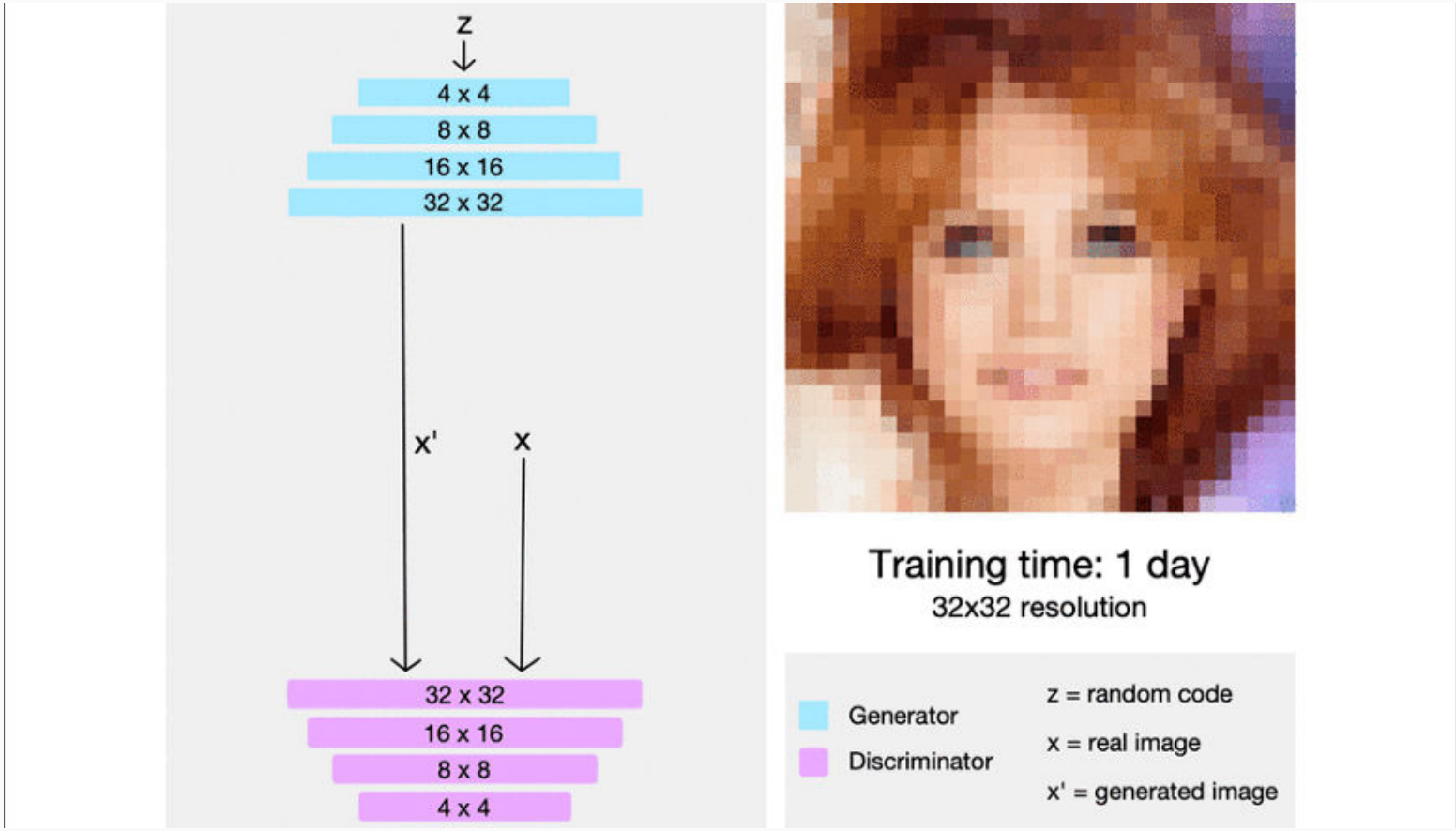
ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024



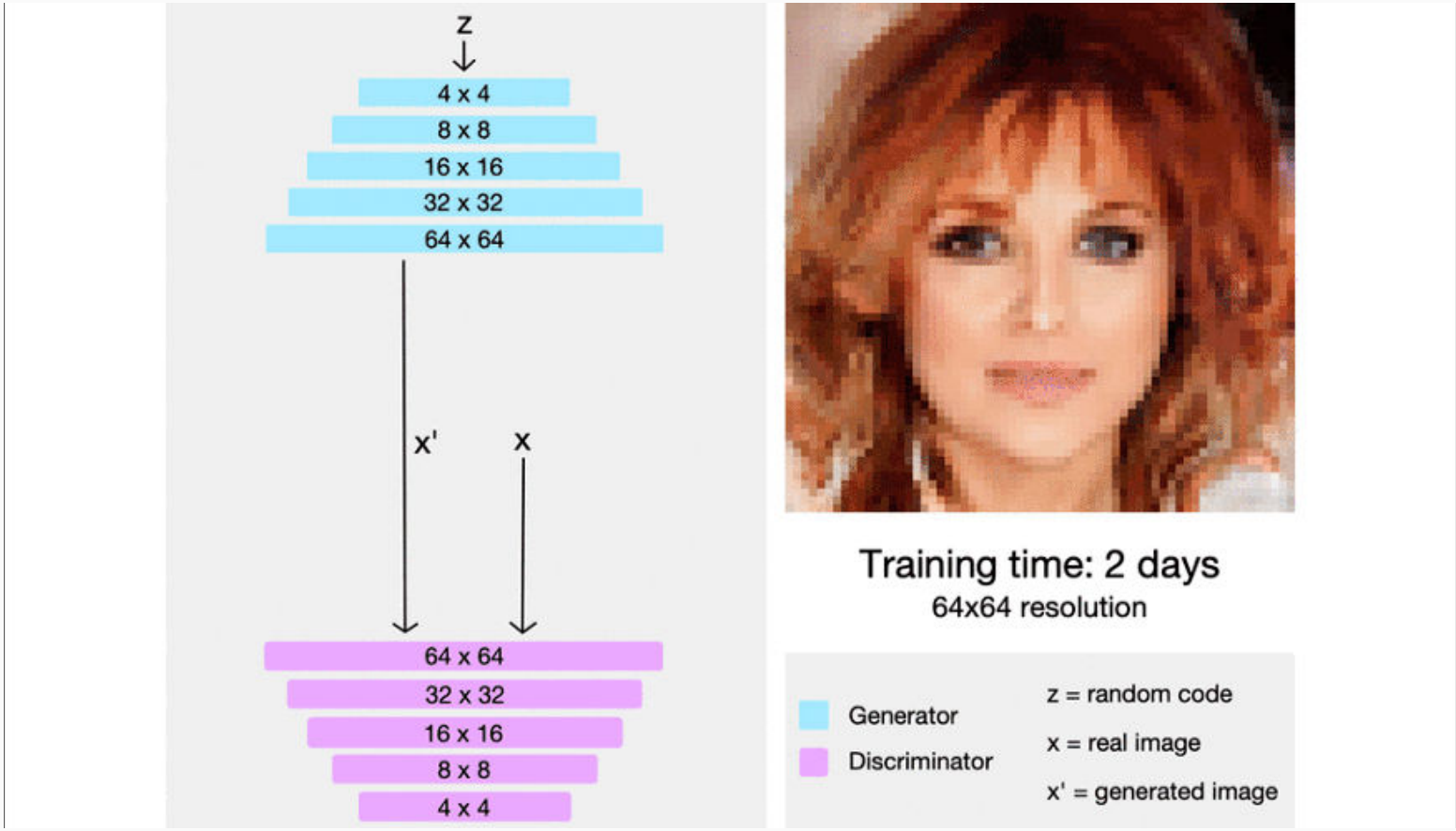
ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024



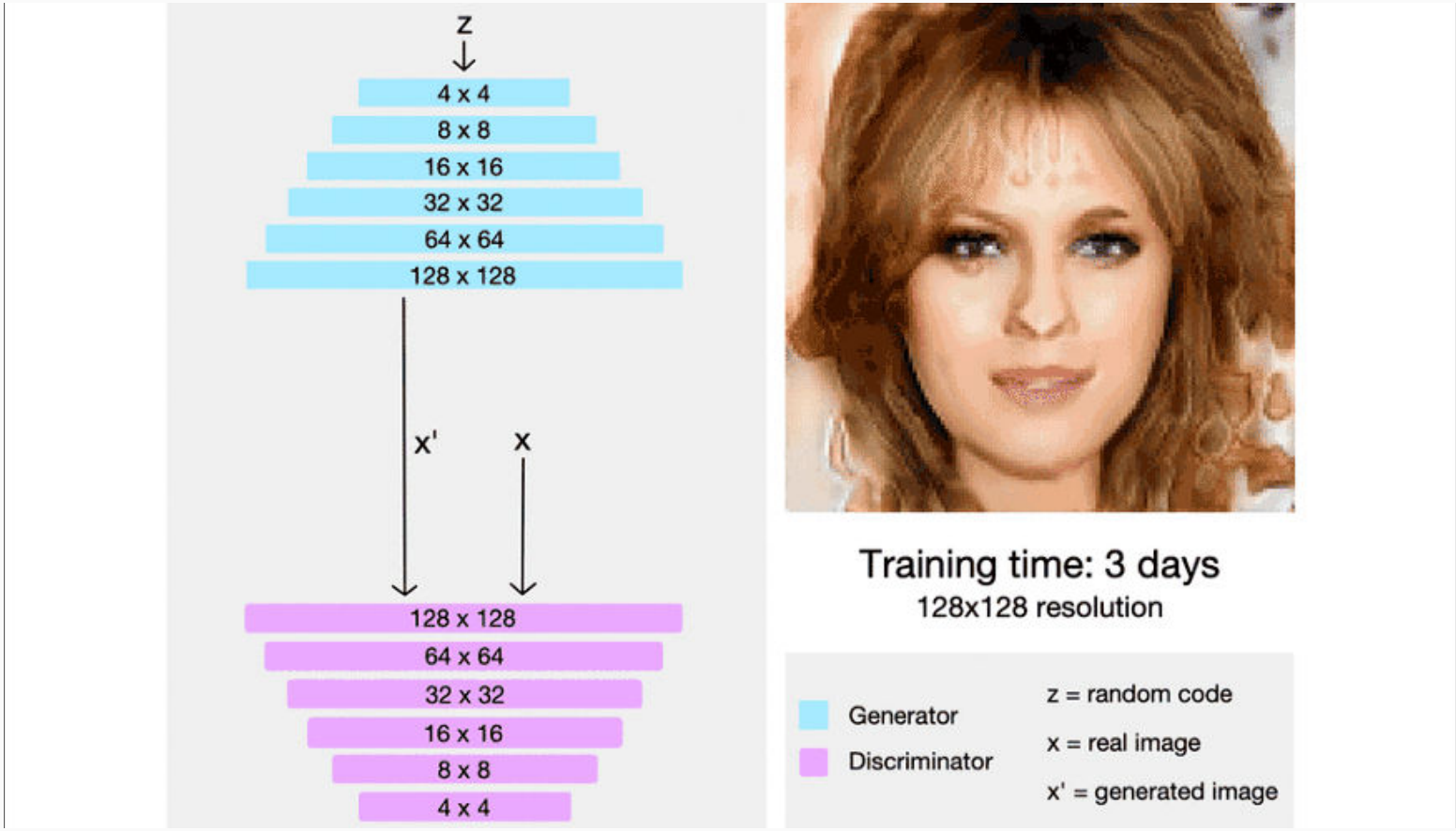
ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024



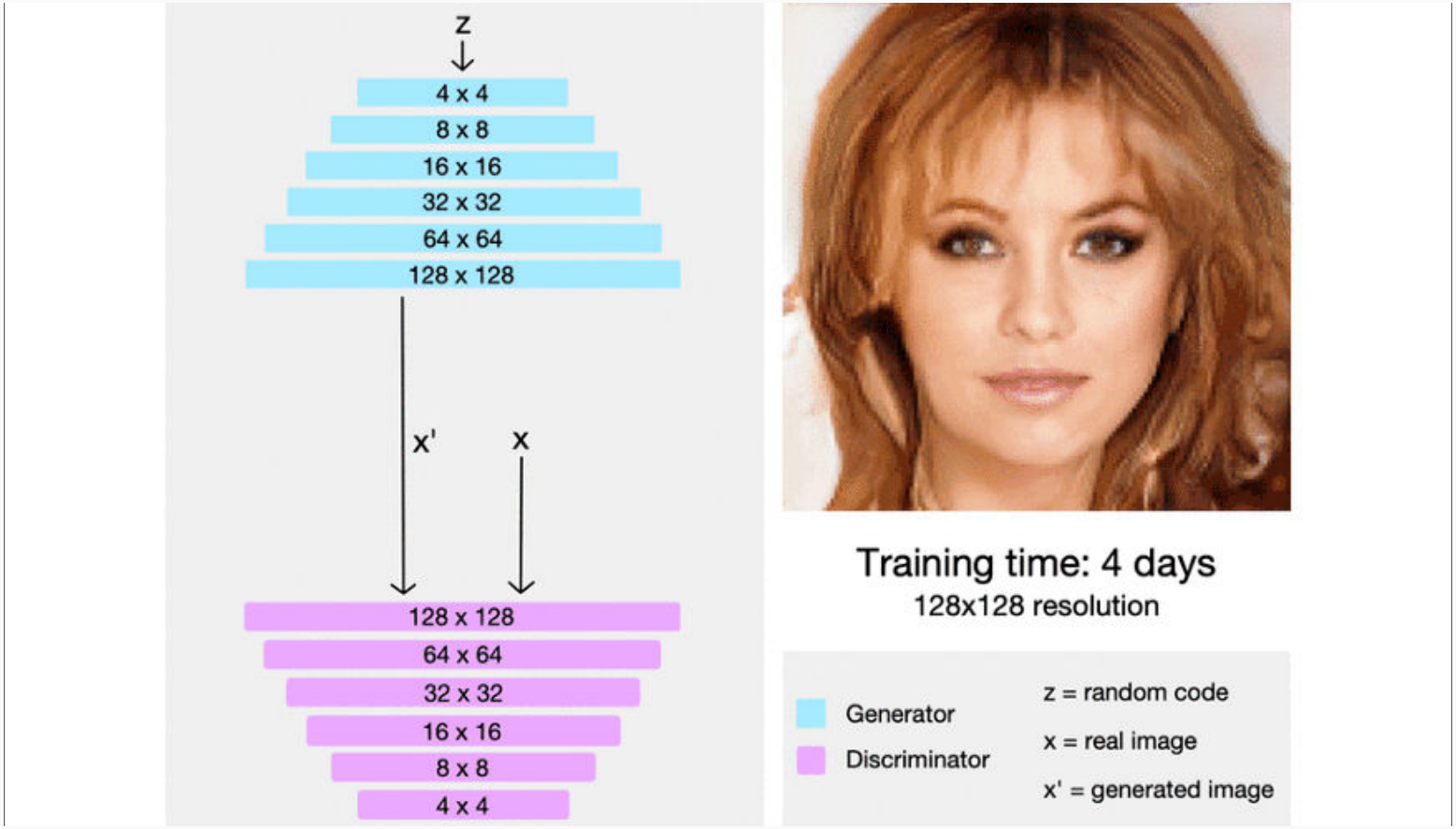
ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024



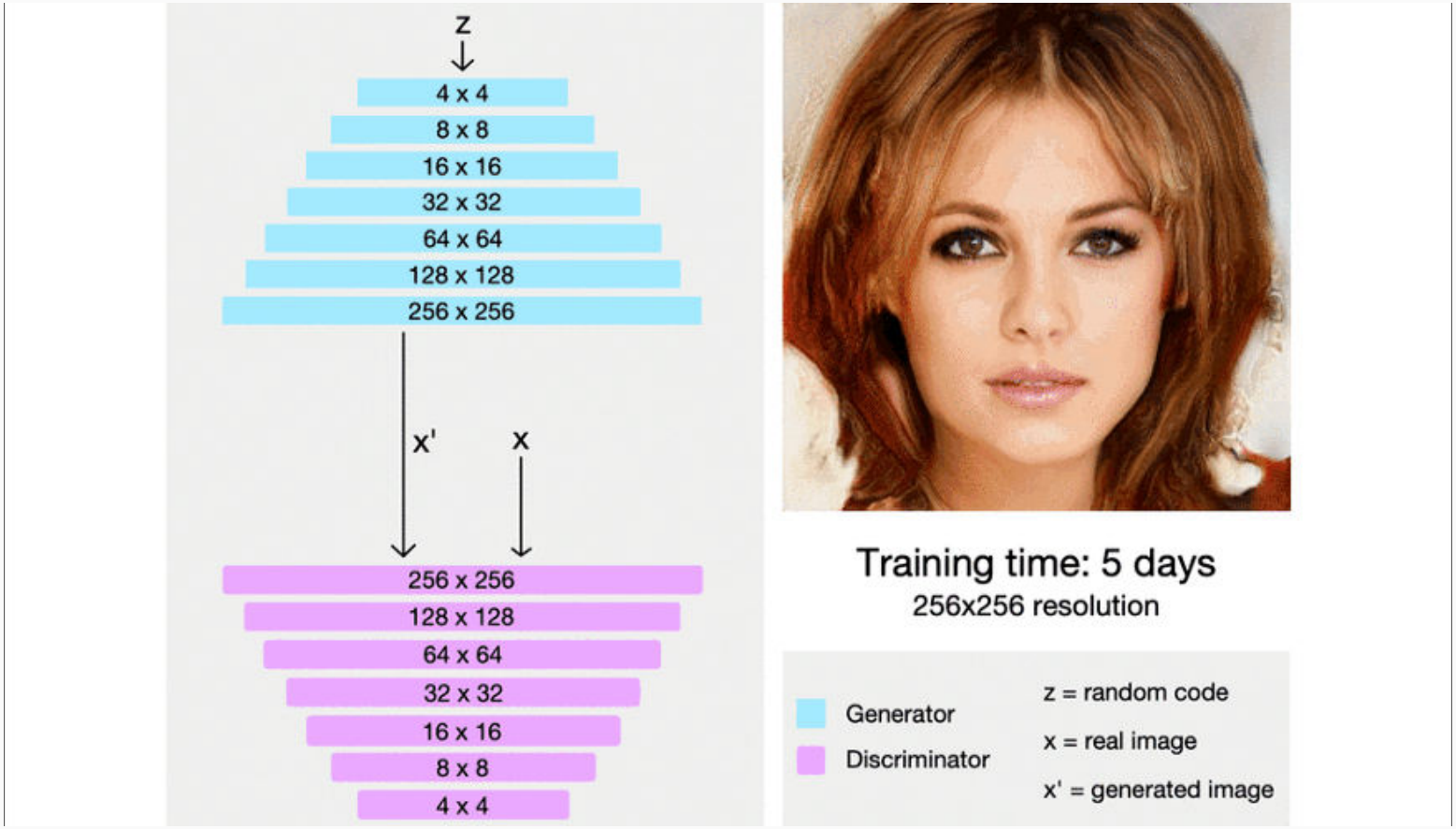
ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024



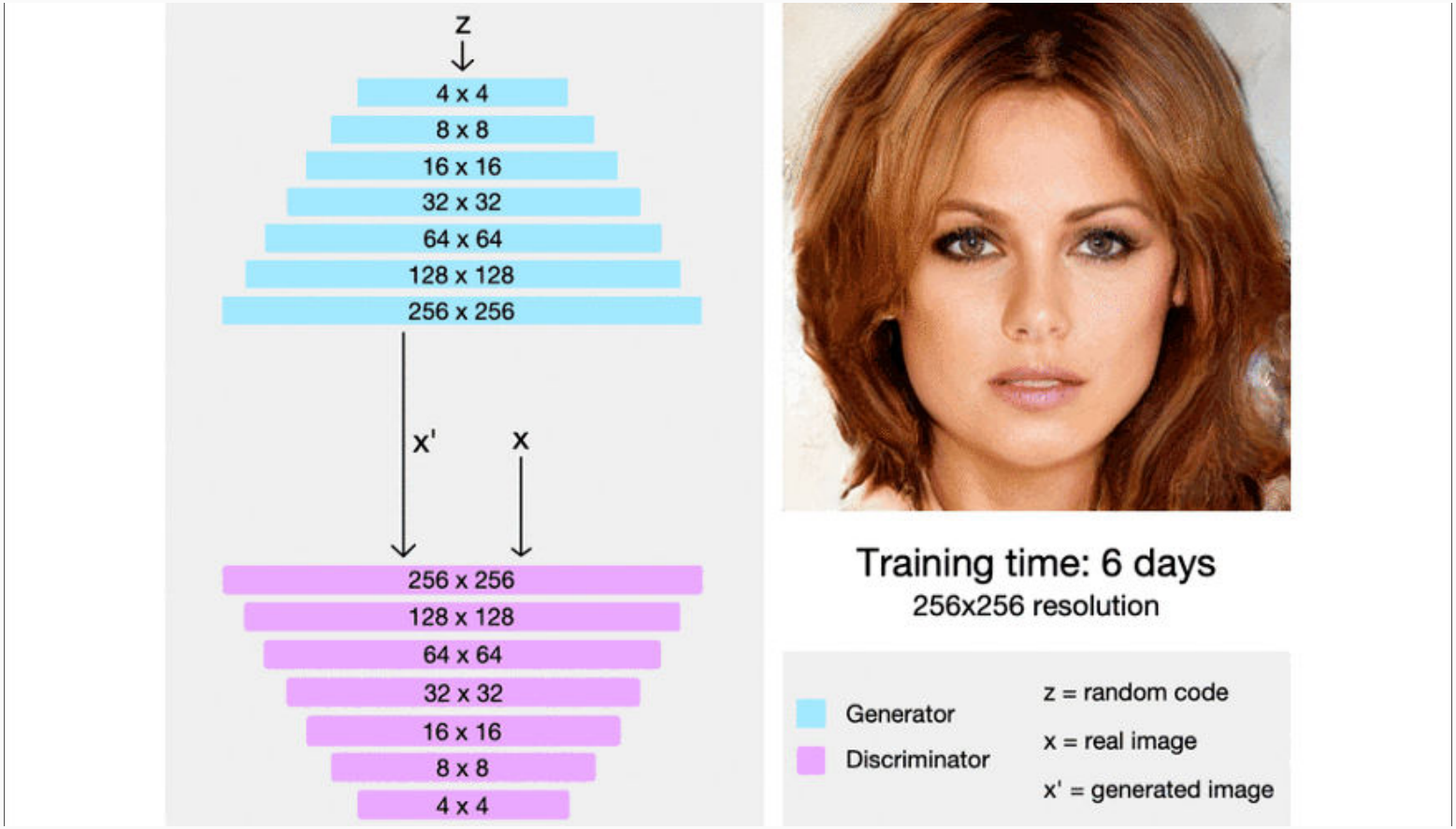
ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024



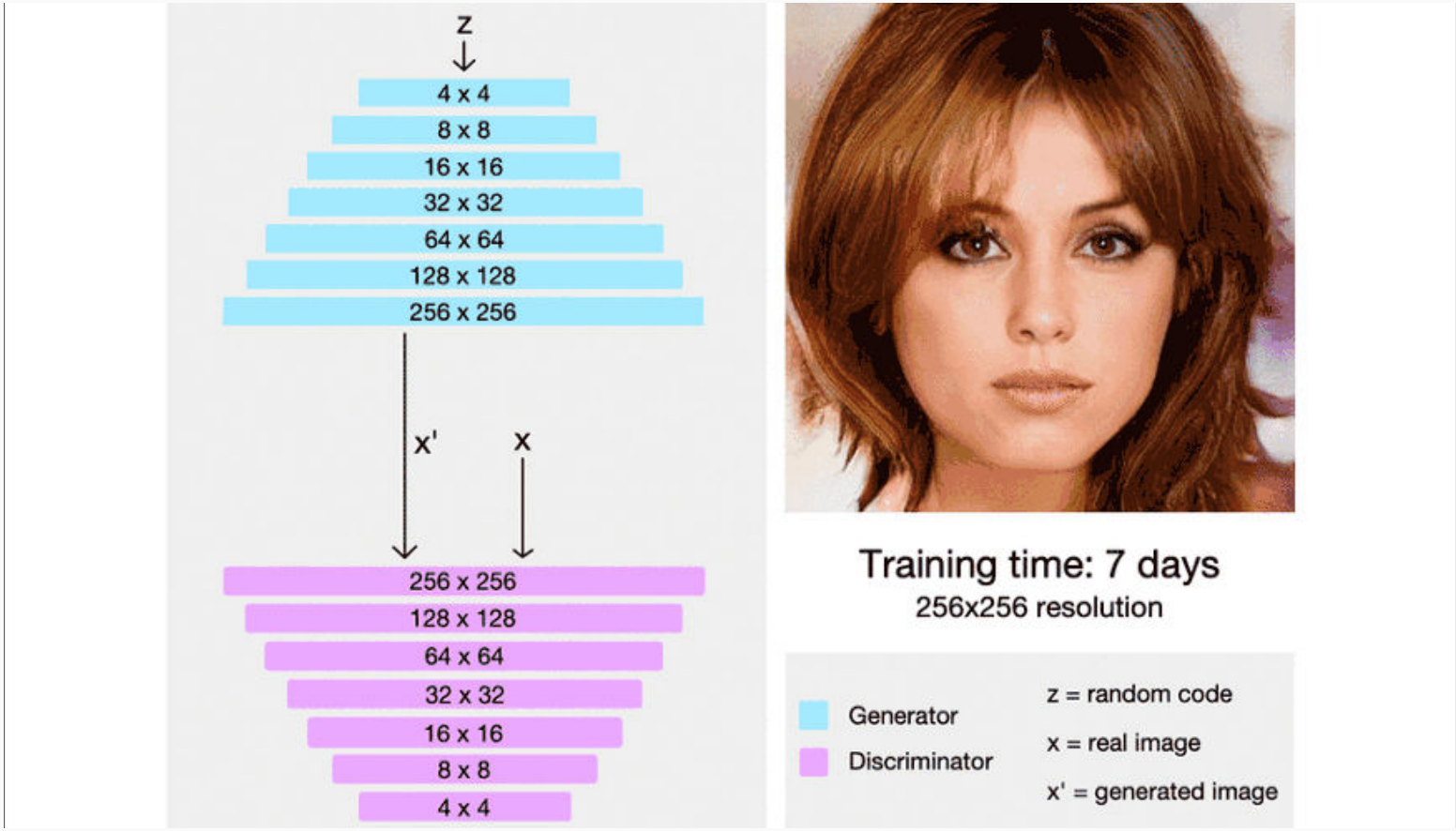
ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024



ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024

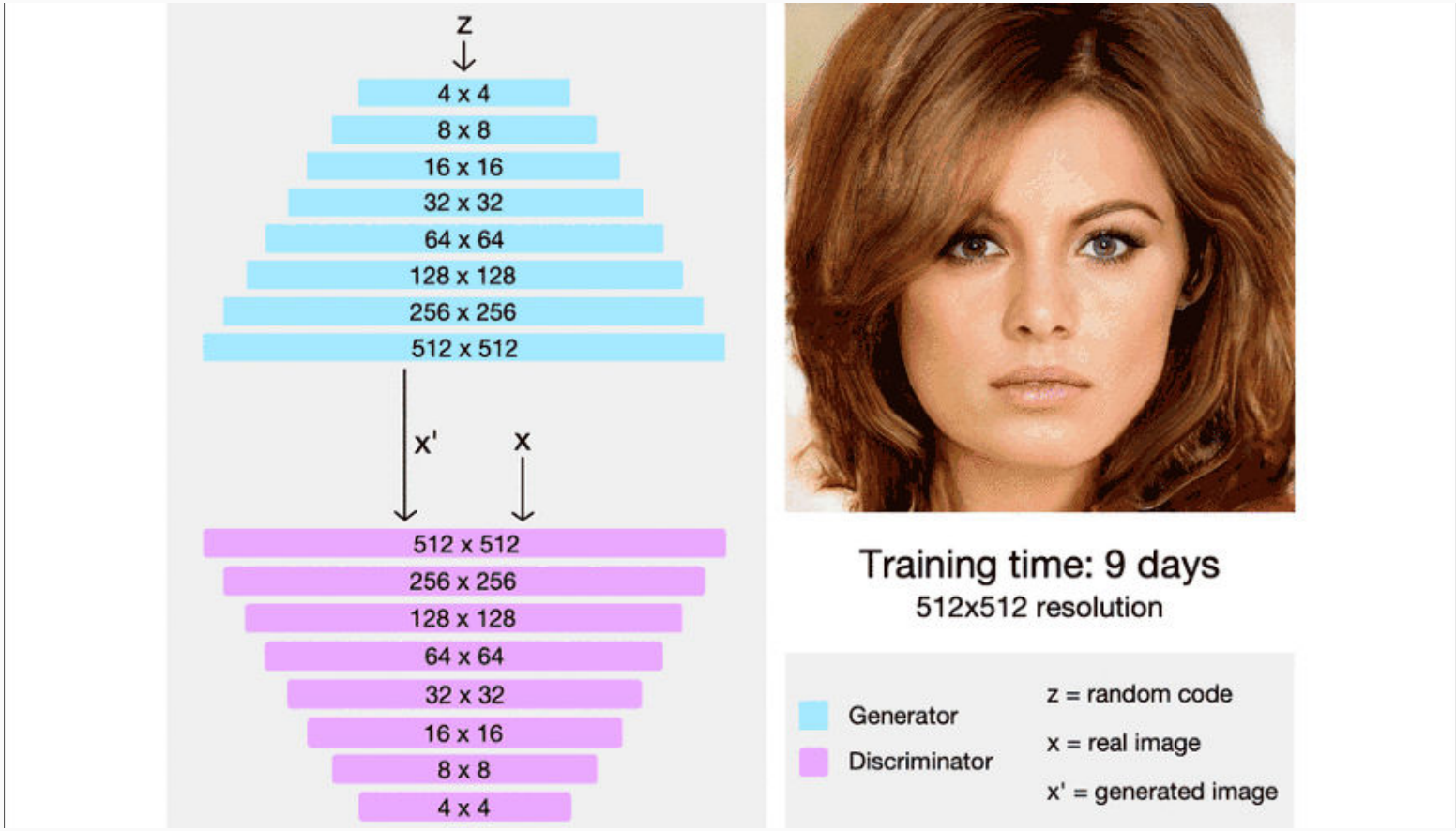


ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024

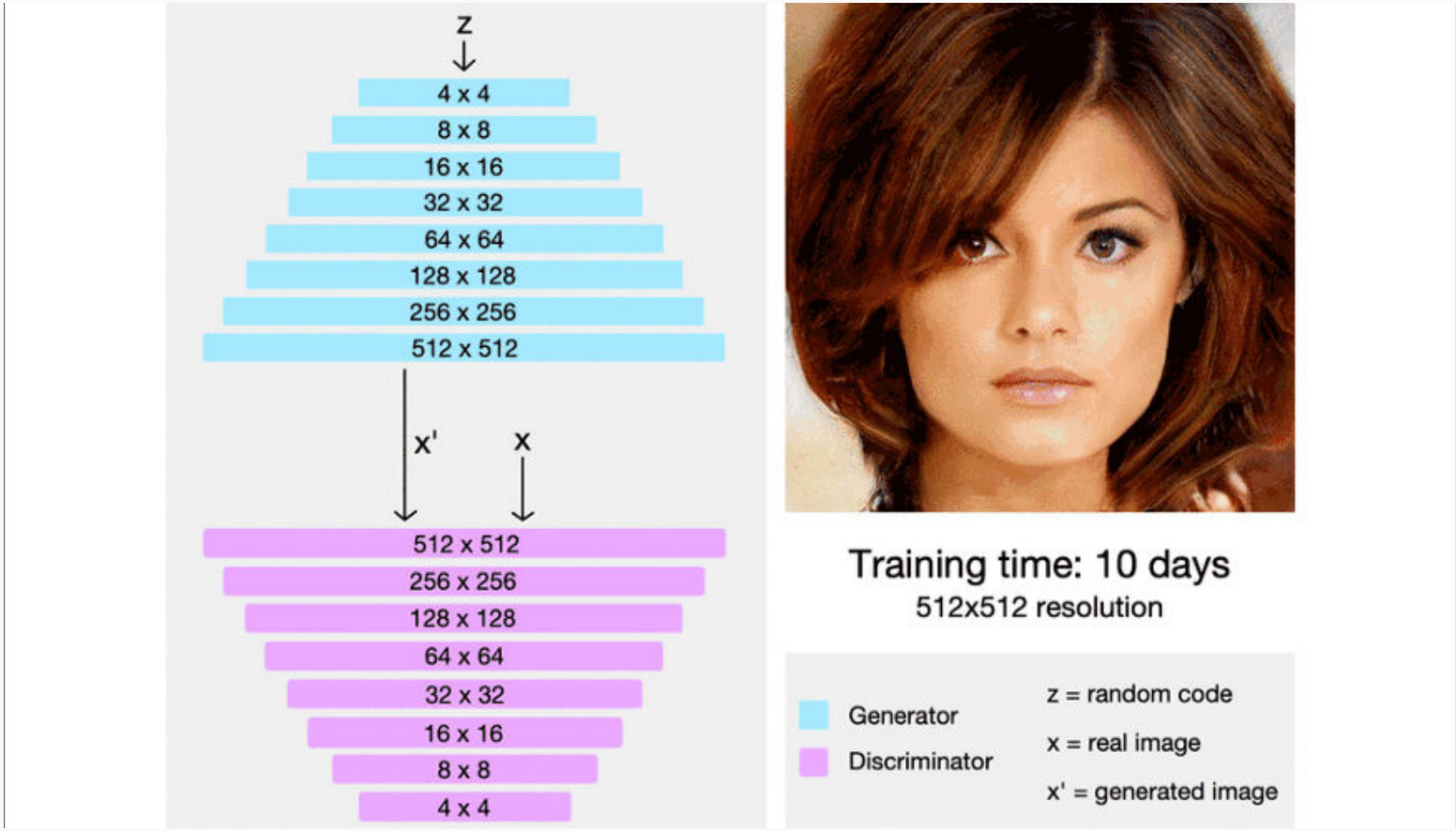
ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024



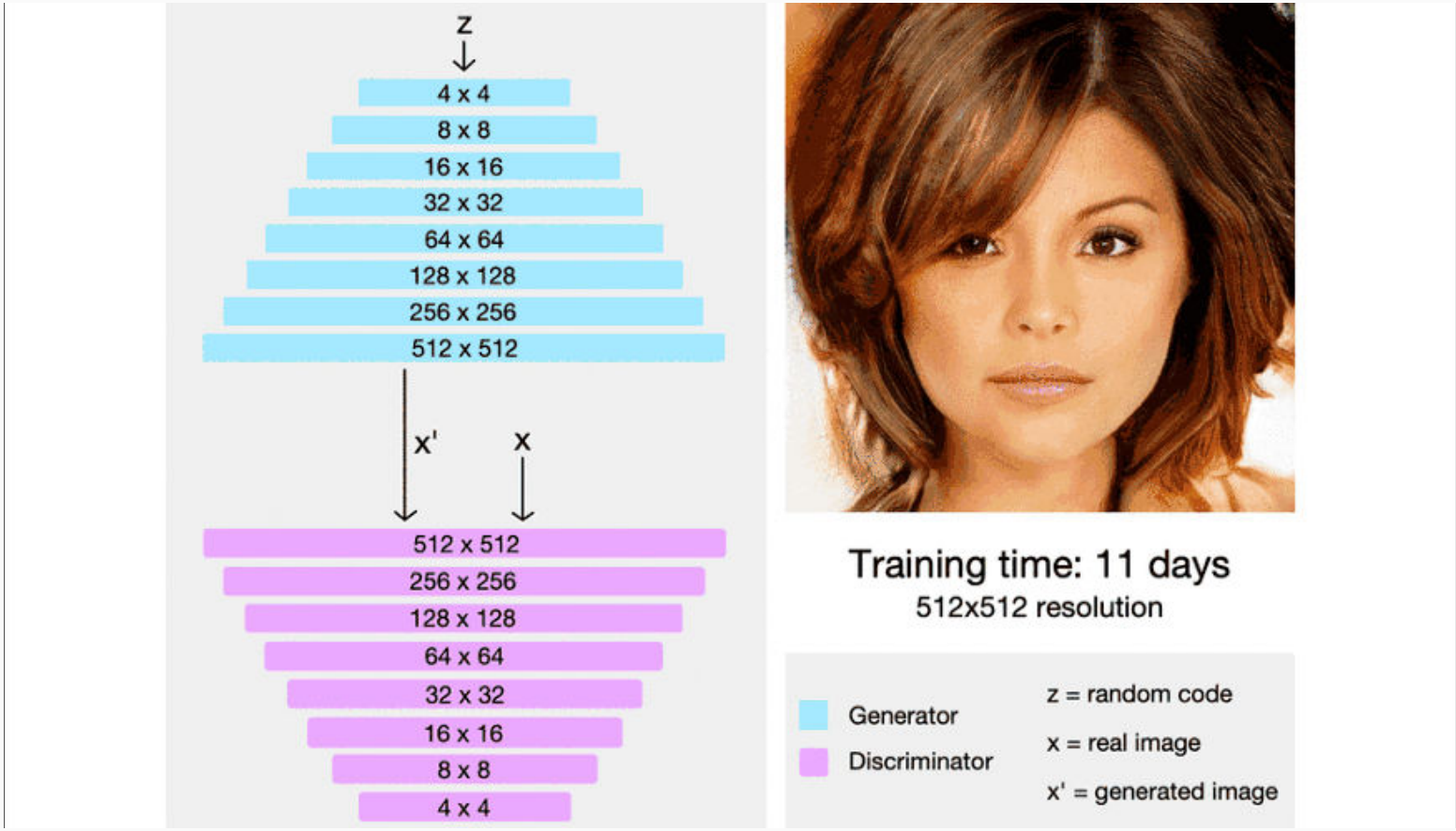
ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024



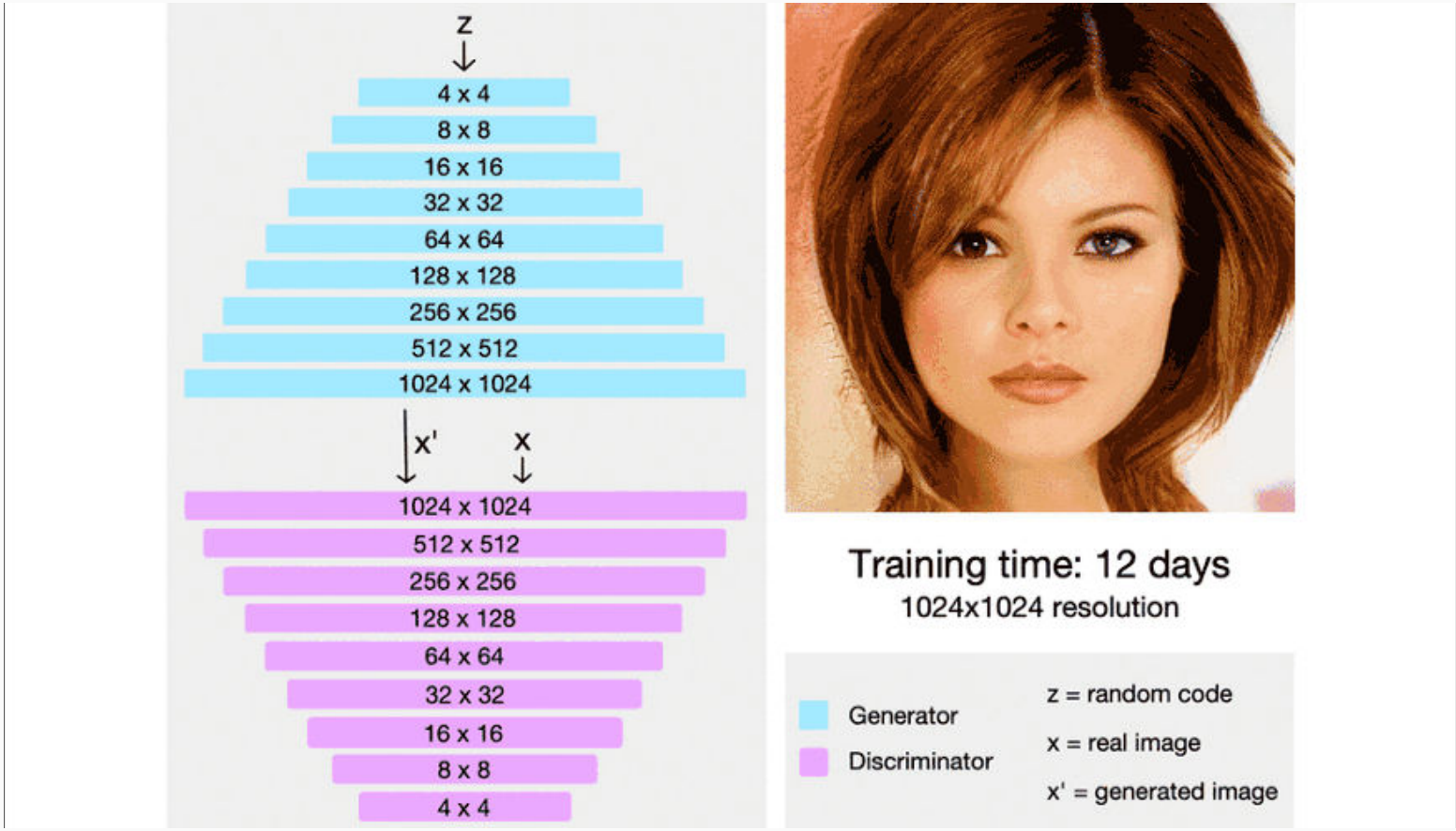
ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024



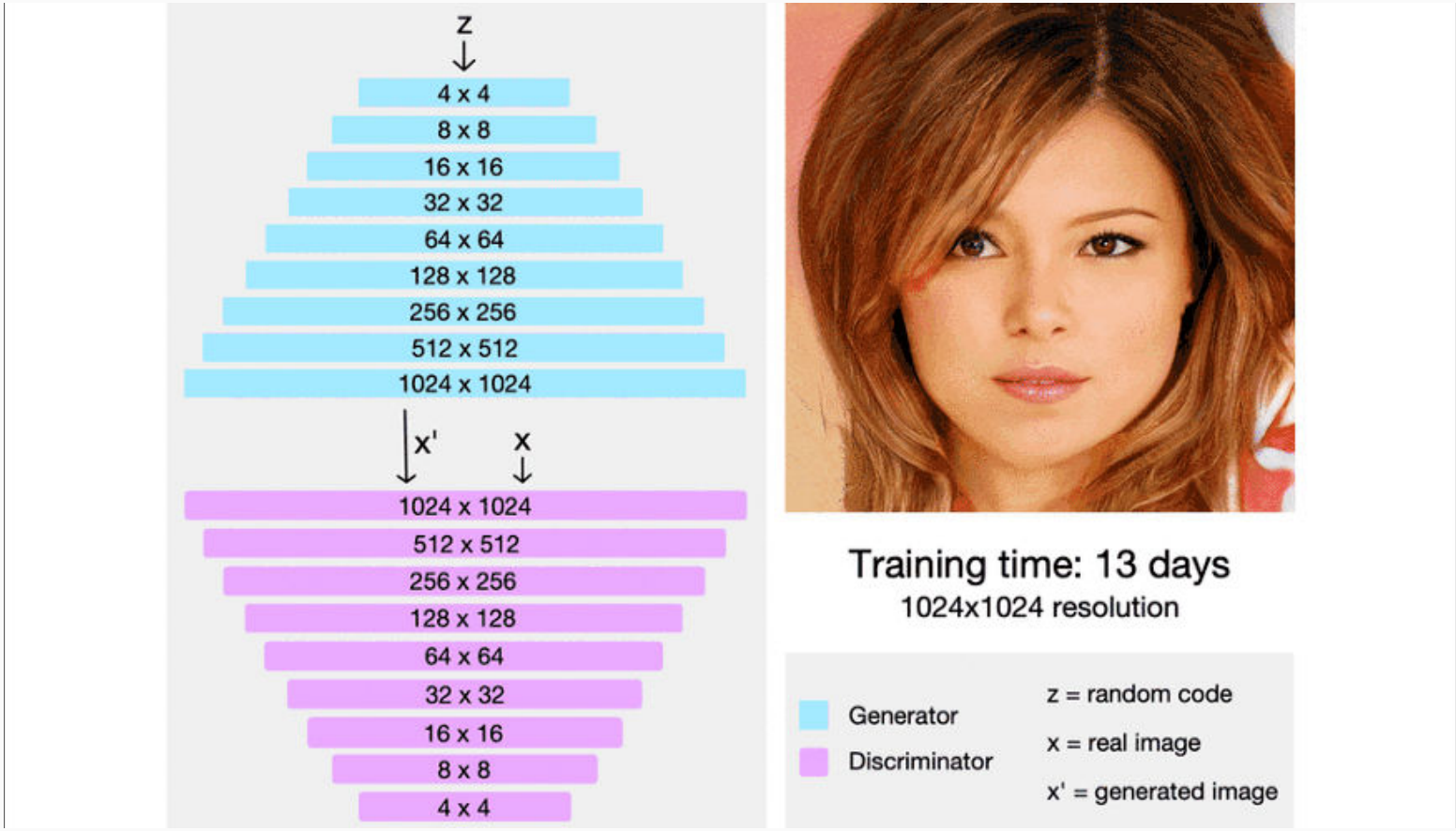
ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024



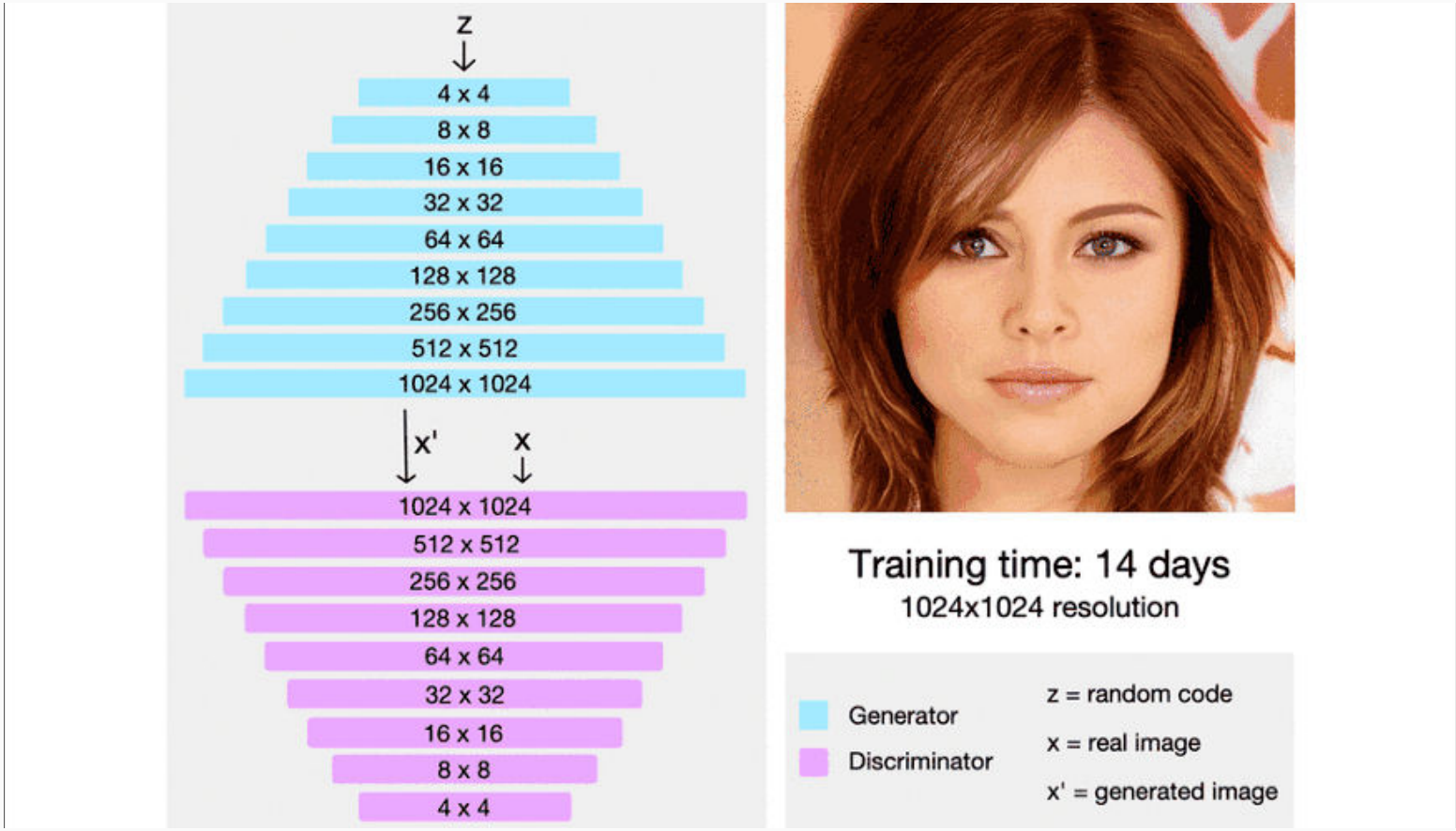
ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024



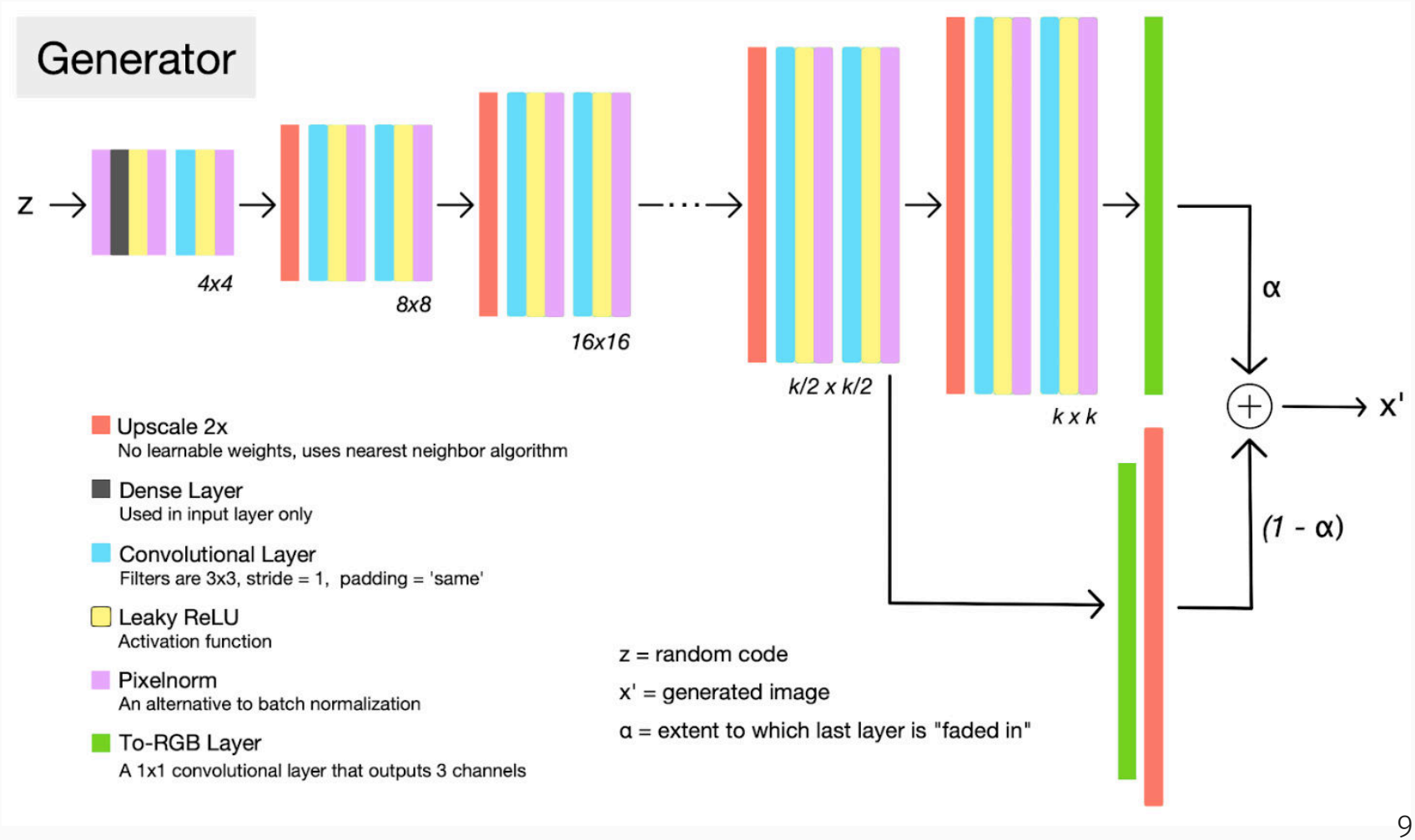
ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024



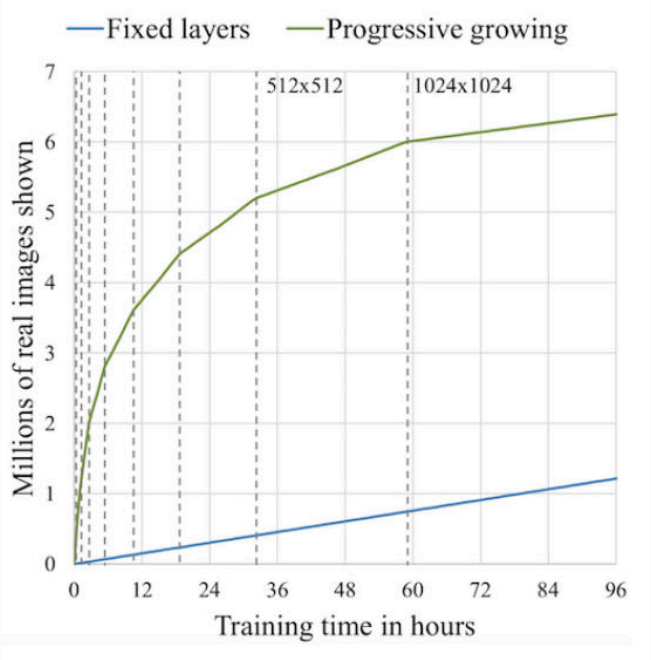
ProGAN

- Именно добавляем новые слои каждый раз; дискриминатор примерно симметричен



ProGAN

- Это помогает экономить время обучения, показывать больше данных за то же время



- Давайте посмотрим:
  - `https://www.youtube.com/watch?v=G06dEcZ-QTg`
  - `https://www.youtube.com/watch?v=36lE9tV9vm0`
- На других категориях до 1024×1024 не доводили, но 256×256 выглядит неплохо

ProGAN

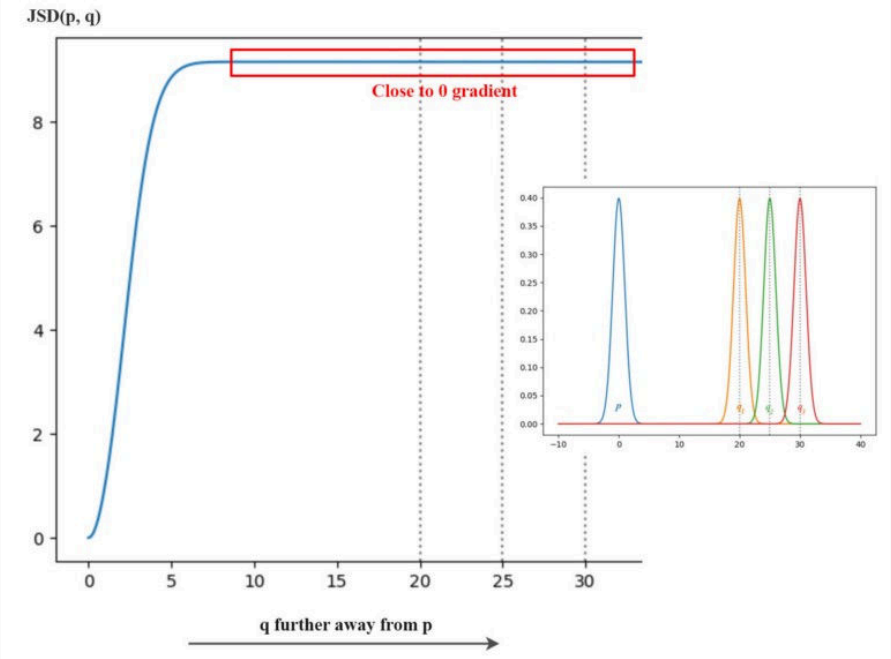
- На других категориях до 1024×1024 не доводили, но 256×256 выглядит неплохо

- И всё это обучается довольно стандартным способом: авторы сравнивают функции ошибки от least squares GAN и от Wasserstein GAN...
- ...wait, what??? Давайте разбираться...

Функции ошибки в GAN

LSGAN

- (Mao et al., 2016): Least Squares GAN (LSGAN)
- Проблема с обычными GAN'ами: функция ошибки (дивергенция Йенсена-Шеннона) насыщается, когда распределение генератора далеко от правильного, и ничего не обучается.



- Это потому, что дискриминатор в обычном GAN работает как классификатор с логистическим сигмоидом.

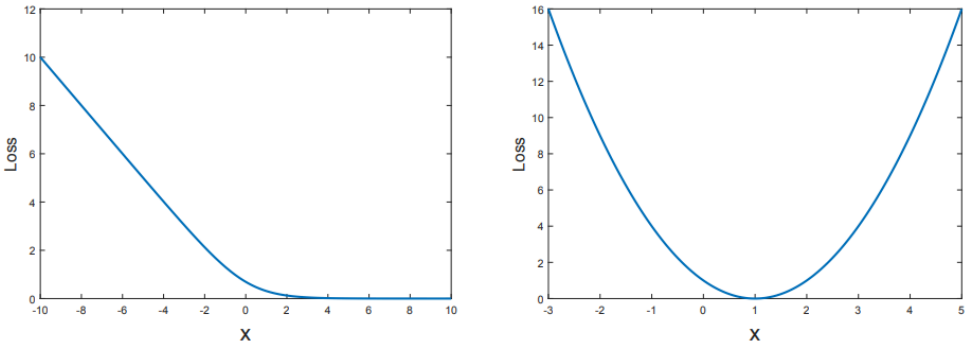
LSGAN

- Давайте попробуем перейти от сигмоидальной к квадратичной функции ошибки:

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

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

т.е. дискриминатор учится отвечать a для фейков и b для настоящих данных, а генератор хочет убедить его отвечать c на фейках. Обычно берут просто $a = 0, b = c = 1$.

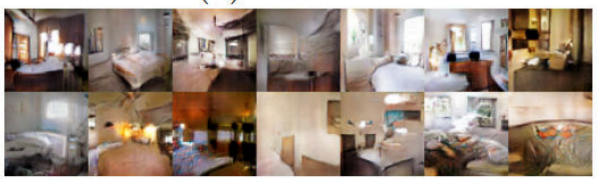


LSGAN

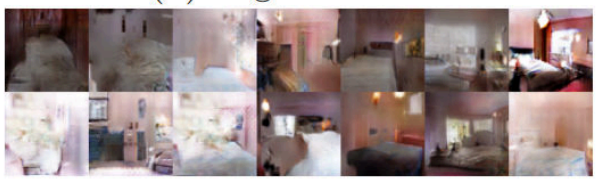
- Mao et al. показали, что LSGAN устойчивее к изменениям архитектуры, меньше страдает от mode collapse; в целом, сейчас квадратичная функция ошибки применяется часто.

(a) LSGANs.

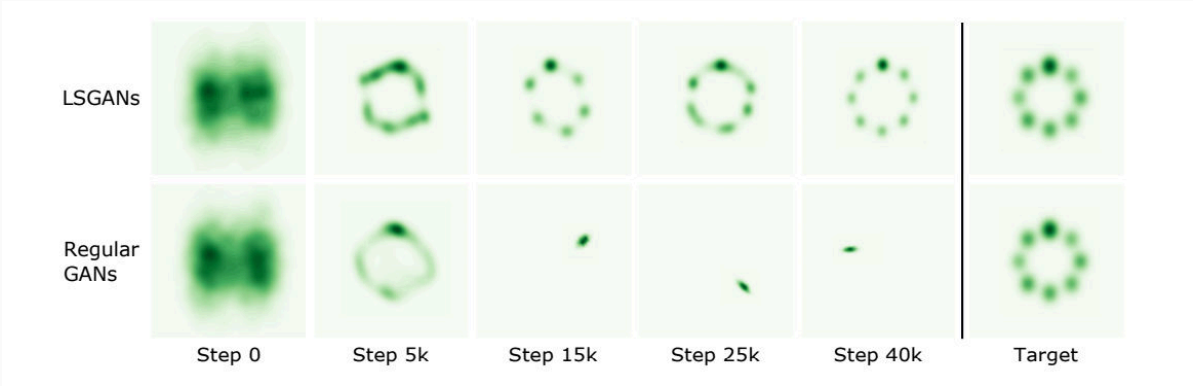
(b) Regular GANs.



(c) LSGANs.



(d) Regular GANs.



InfoGAN

- (Chen et al., 2016): InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets
- Важное дополнение: было бы круто добавить disentanglement, чтобы разные признаки имели смысл
- В обычных GAN'ах используют просто вектор шума $\mathbf{z}$. В InfoGAN мы его разбиваем на части:
  - $\mathbf{z}$ – это действительно несжимаемый шум, источник энтропии;
  - $\mathbf{c} = c_1 c_2 \dots c_L$ – это латентный код.
- Можно в обычный GAN попробовать это добавить, чтобы генератор стал $G(\mathbf{z}, \mathbf{c})$, но надо что-то добавить, чтобы $\mathbf{c}$ было важно и G не мог бы просто забить на него.

InfoGAN

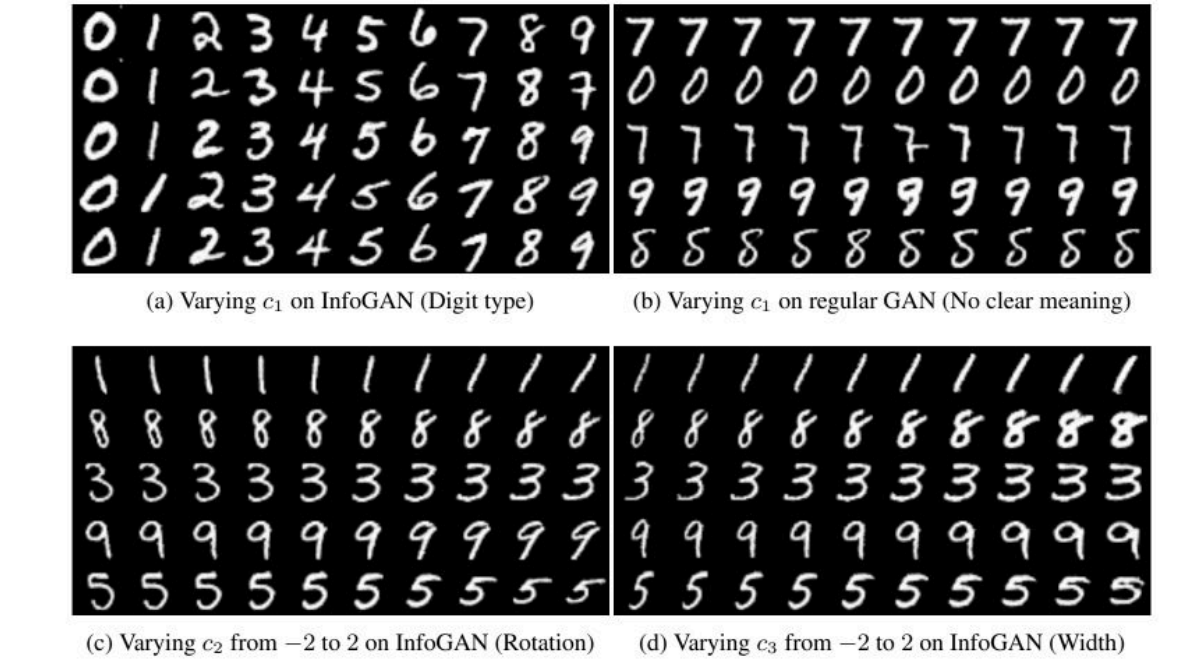
- InfoGAN: будем требовать, чтобы была большая взаимная информация (mutual information) $I(\mathbf{c}, G(\mathbf{z}, \mathbf{c}))$, т.е. в распределении $G(\mathbf{z}, \mathbf{c})$ должно остаться много от $\mathbf{c}$.
- Формально – вариационная оценка:

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

- И эту нижнюю оценку мы через reparametrization trick параметризуем нейросетью Q; она обычно почти целиком совпадает с D.

InfoGAN

- Пример на MNIST с $\mathbf{c} = c_1 c_2 c_3$, где c_1 – дискретная метка с 10 категориями (без supervision! с равномерным априорным распределением), и $c_2, c_3 \sim U[-1, 1]$:



(a) Varying c_1 on InfoGAN (Digit type)

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

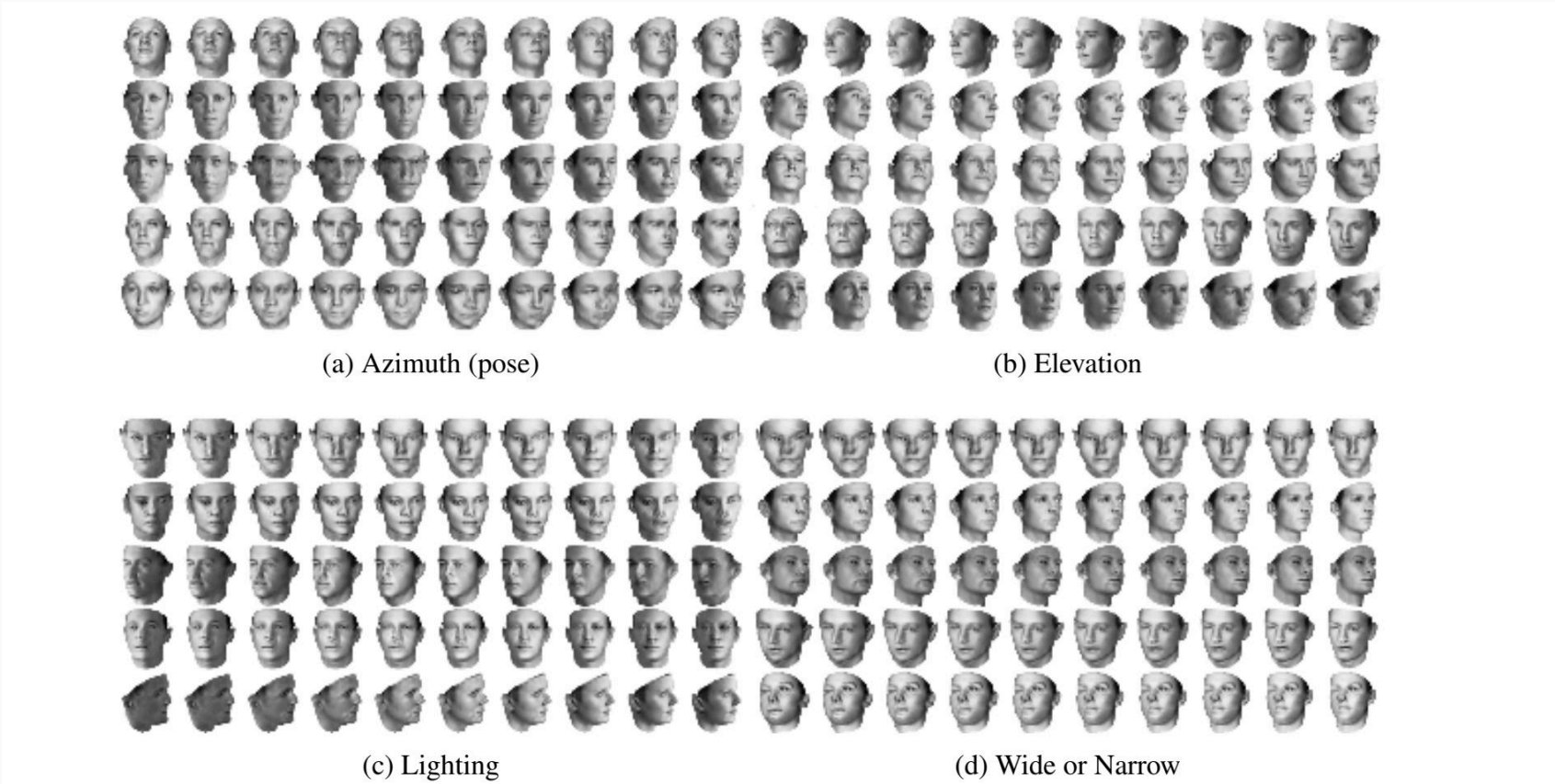
(c) Varying c_2 from -2 to 2 on InfoGAN (Rotation)

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

- На самом деле по коду c_1 классификатор даёт ошибку 5% на MNIST без всякого supervision.

InfoGAN

- 3D-лица:



InfoGAN

- 3D-стулья:

Wasserstein GAN

- Wasserstein GAN (Arjovsky et al., 2017):
  - что такое обучить распределение вероятностей?
  - это значит минимизировать $\mathrm{KL}(p_{\text{data}} \| p_{\text{model}})$;
  - но для этого нужно, чтобы p_{model} существовала, а это неверно, если распределение сосредоточено на подмногообразиях малой размерности; вряд ли многообразие p_{model} будет существенно пересекаться с многообразием p_{data}, и KL будет не определено (бесконечно);
  - обычно мы добавляем шум (гауссовский, например), все модели в ML с шумом;
  - но для порождающих моделей шум мешает, делает порождённые примеры размытыми;
  - поэтому обычно шум используют для подсчёта ML, но убирают во время порождения, т.е. шум – это неправильно, но надо для ML.

Wasserstein GAN

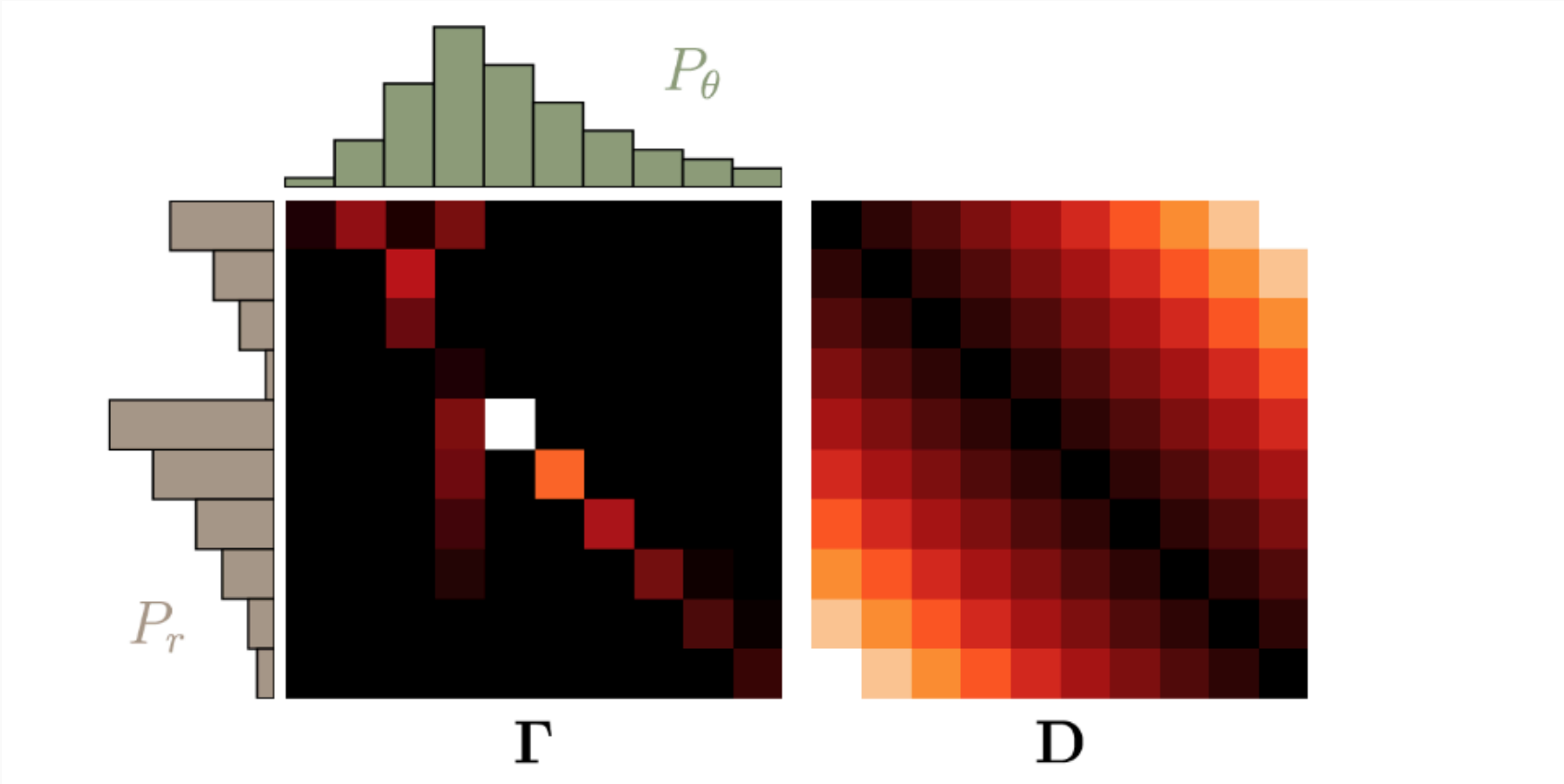
- Wasserstein GAN (Arjovsky et al., 2017): давайте посмотрим на другие метрики близости между p_{data} и p_{model}.
- Earth Mover distance, или Wasserstein-1:

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

где $\Pi(p_{\text{data}}, p_{\text{model}})$ – множество совместных распределений $\gamma(\mathbf{x}, \mathbf{y})$, маргиналы которых – p_{data} и p_{model}.

Wasserstein GAN

- Т.е. $\gamma(\mathbf{x}, \mathbf{y})$ показывает, сколько надо «массы» перераспределить от $\mathbf{x}$ к $\mathbf{y}$, чтобы превратить p_{data} в p_{model}.



Wasserstein GAN

- Пример: рассмотрим $Z \sim U[0,1]$, $\mathbb{P}_0$ – распределение $(0, Z)$ на $\mathbb{R}^2$, и $g_\theta(z) = (\theta, z)$, где θ – параметр. Тогда

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

- Т.е. только EM-расстояние действительно куда-то сходится при $\theta \to 0$, а это ведь очень похоже на обучение распределения, сосредоточенного на подмногообразии.

Wasserstein GAN

- Можно доказать, что $W(p_{\text{data}}, p_{\text{model}})$ – непрерывная функция от параметров θ распределения p_{model} при достаточно слабых условиях.
- А двойственность Монжа-Канторовича (Kantorovich-Rubinstein duality) говорит, что инфимум

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

эквивалентен

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

где супремум берётся по всем функциям с липшицевой константой ≤ 1.
- А мы хотим обучить порождающую модель $p_{\text{model}} = g_\theta(\mathbf{z})$.
- Давайте всё параметризуем нейросетями.

Wasserstein GAN

- Сеть $f_{\mathbf{w}}$ для функции f и сеть для g_θ. Тогда:
  - для данного g_θ обучим веса $f_{\mathbf{w}}$, максимизируя

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

  - для данного $f_{\mathbf{w}}$ подсчитаем

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

- А чтобы $f_{\mathbf{w}}$ оставалось липшицевым, можно просто weight clipping сделать для градиентов.

Wasserstein GAN

- Итого алгоритм обучения:

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

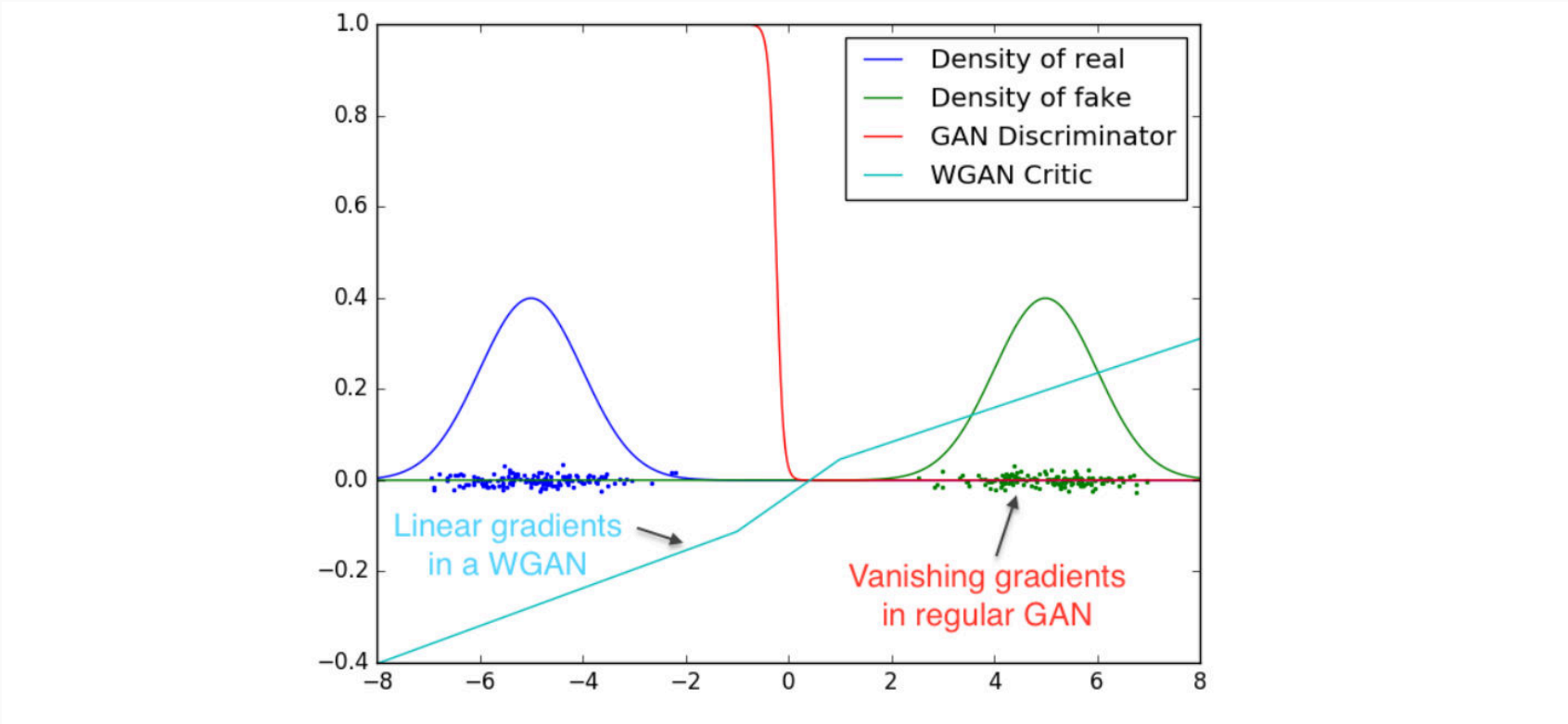
Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

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

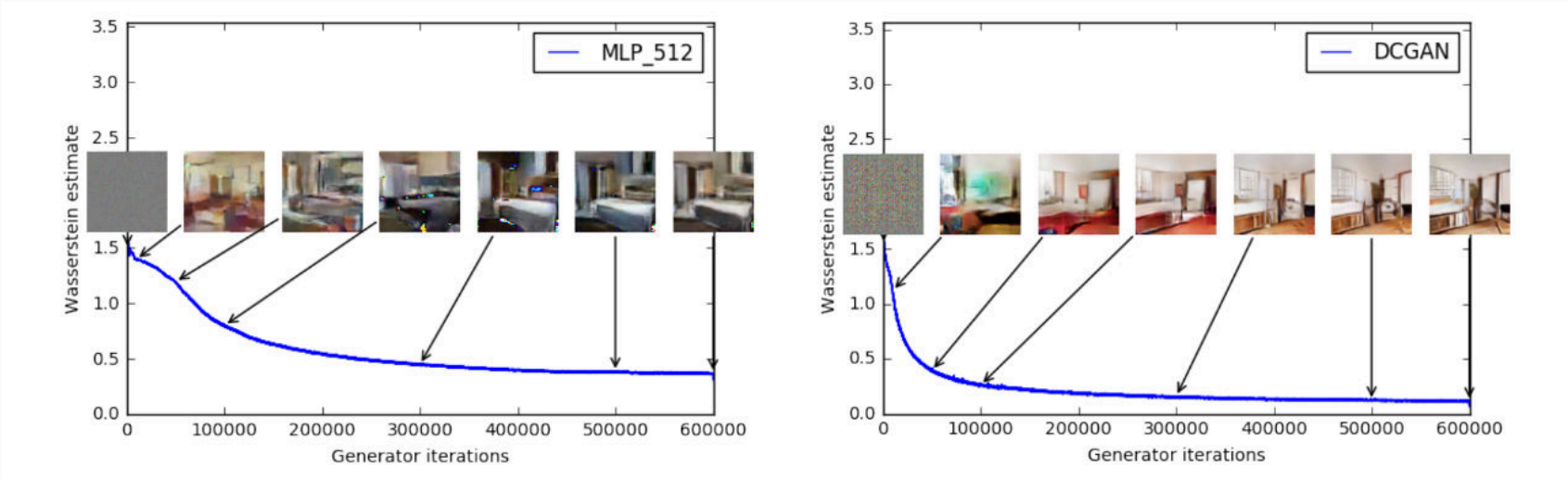
Wasserstein GAN

- Пример, на котором видно, что в WGAN можно спокойно обучать $f_{\mathbf{w}}$ до сходимости, а в обычном GAN дискриминатор, может, и не стоит так обучать:



Wasserstein GAN

- Качество тесно коррелирует с функцией ошибки; слева тут просто полносвязная сеть в качестве генератора, справа – DCGAN:

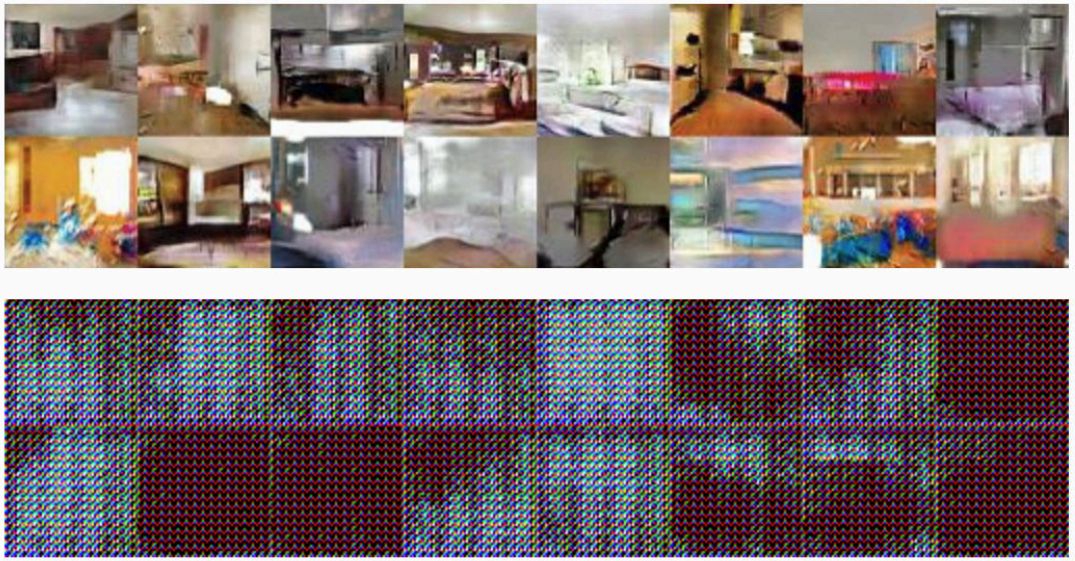


Wasserstein GAN

- WGAN устойчивее к изменению архитектуры; вот обычные WGAN и DCGAN (снизу):

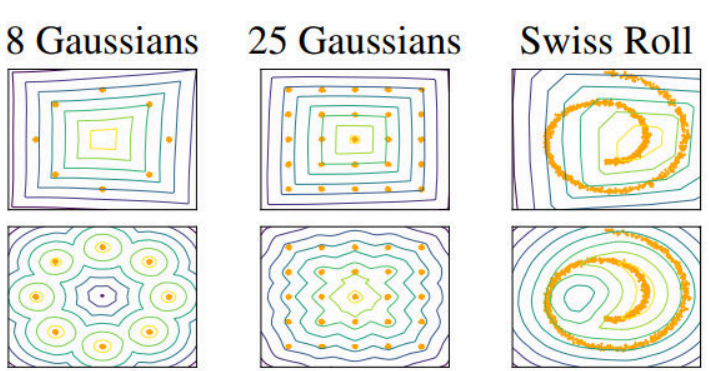
Wasserstein GAN

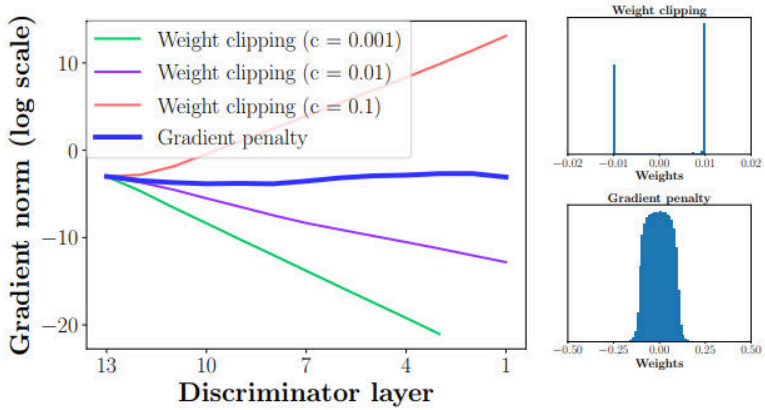
- А вот что будет, если убрать из генератора batchnorm и сделать фильтров поменьше (одинаковое число на каждом уровне):



Wasserstein GAN

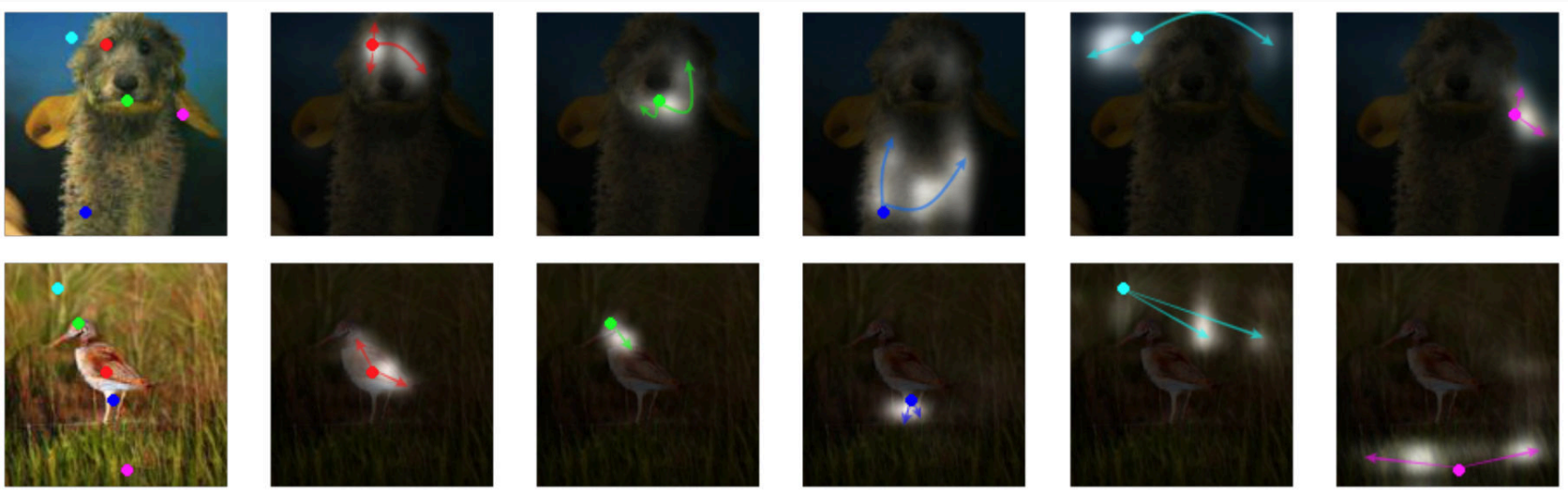
- (Gulrajani et al., 2017): Improved Training of Wasserstein GANs
- Weight clipping в критике на самом деле ведёт к проблемам, не обучаются более высокие моменты распределений и градиенты взрываются/затухают
- Лучше делать мягкий регуляризатор на градиент, типа $\lambda \mathbb{E}_{\hat{\mathbf{x}} \sim P_{\hat{\mathbf{x}}}} \left[(\|\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})\|_2 - 1)^2 \right]$





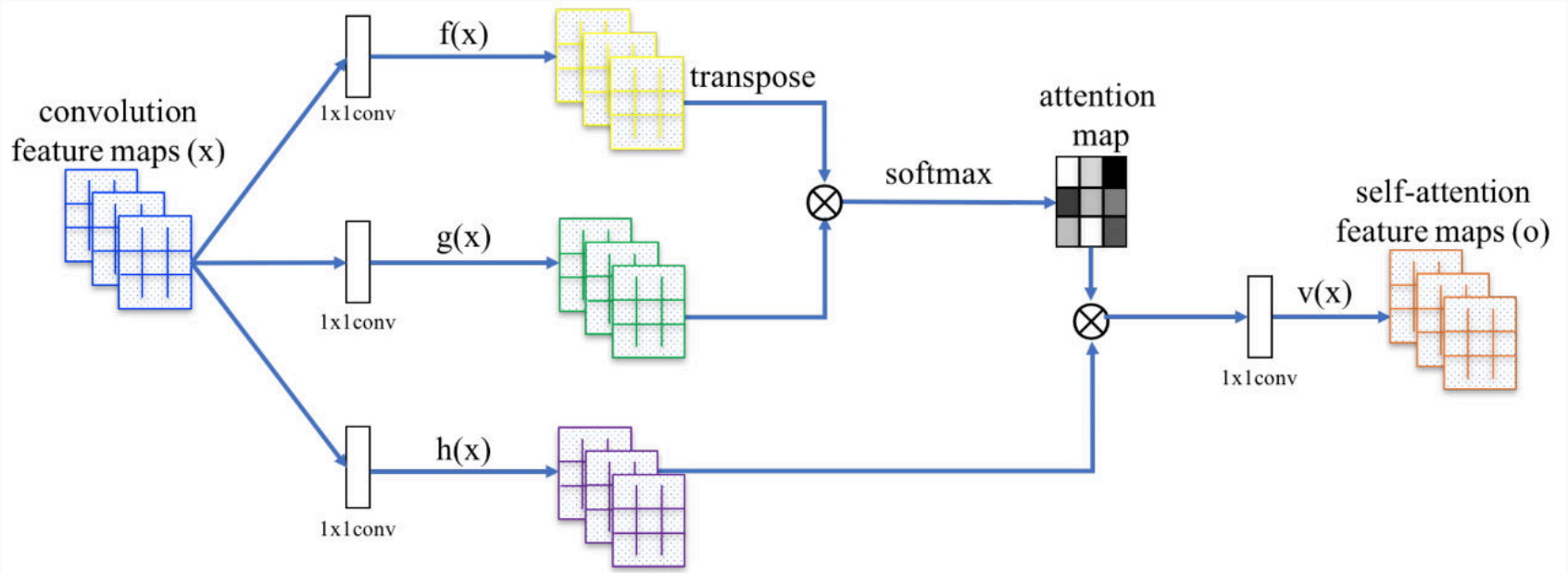
SAGAN

- (Zhang et al., 2019): Self-Attention Generative Adversarial Networks (SAGAN)
- Пытаются решить проблему с локальностью порождения свёрточными сетями



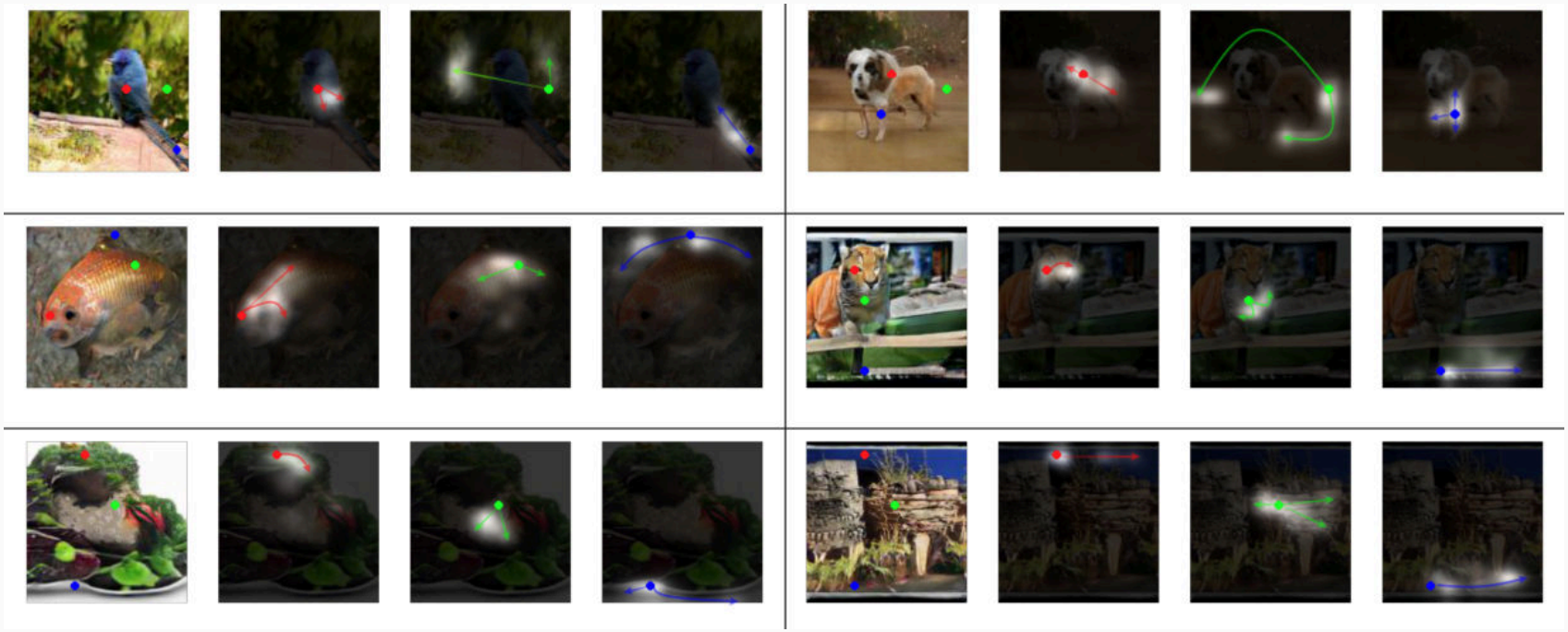
SAGAN

- Self-attention здесь примерно как в Transformer, только переложенный на картинки:



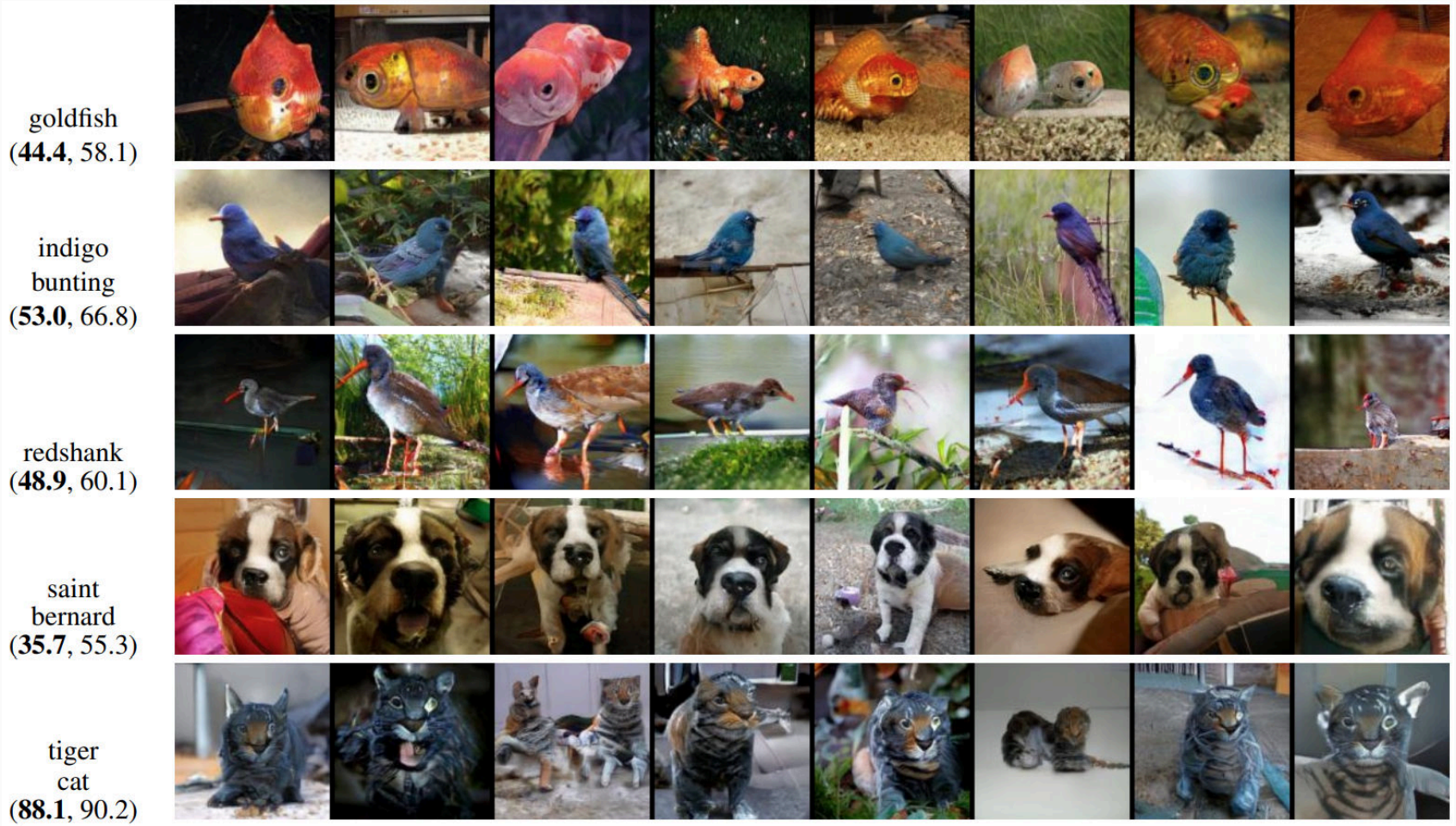
SAGAN

- И получается, что действительно генератор (здесь примеры с последнего слоя) может смотреть достаточно далеко, когда порождает картинку:



SAGAN

- В исходной статье (Zhang et al., 2019) уже получаются хорошие картинки:



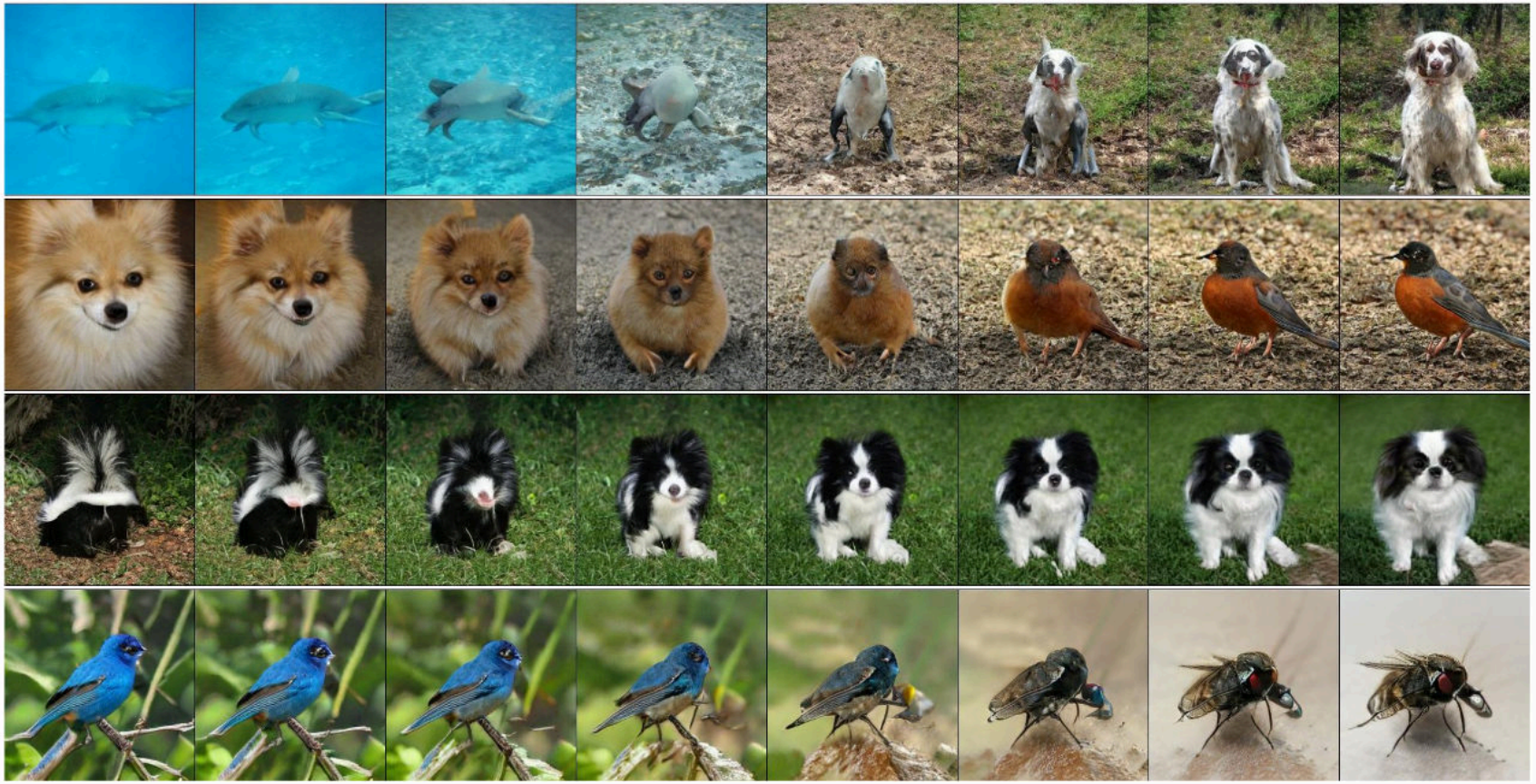
SAGAN

- Следующий этап – BigGAN (Brock et al., 2019), улучшил Inception score в три раза...

- Используют SA-GAN (GAN с self-attention) со всеми новыми трюками и специальными архитектурами

SAGAN

- Интерполяции:



- Ой, а что такое «лучше»?..

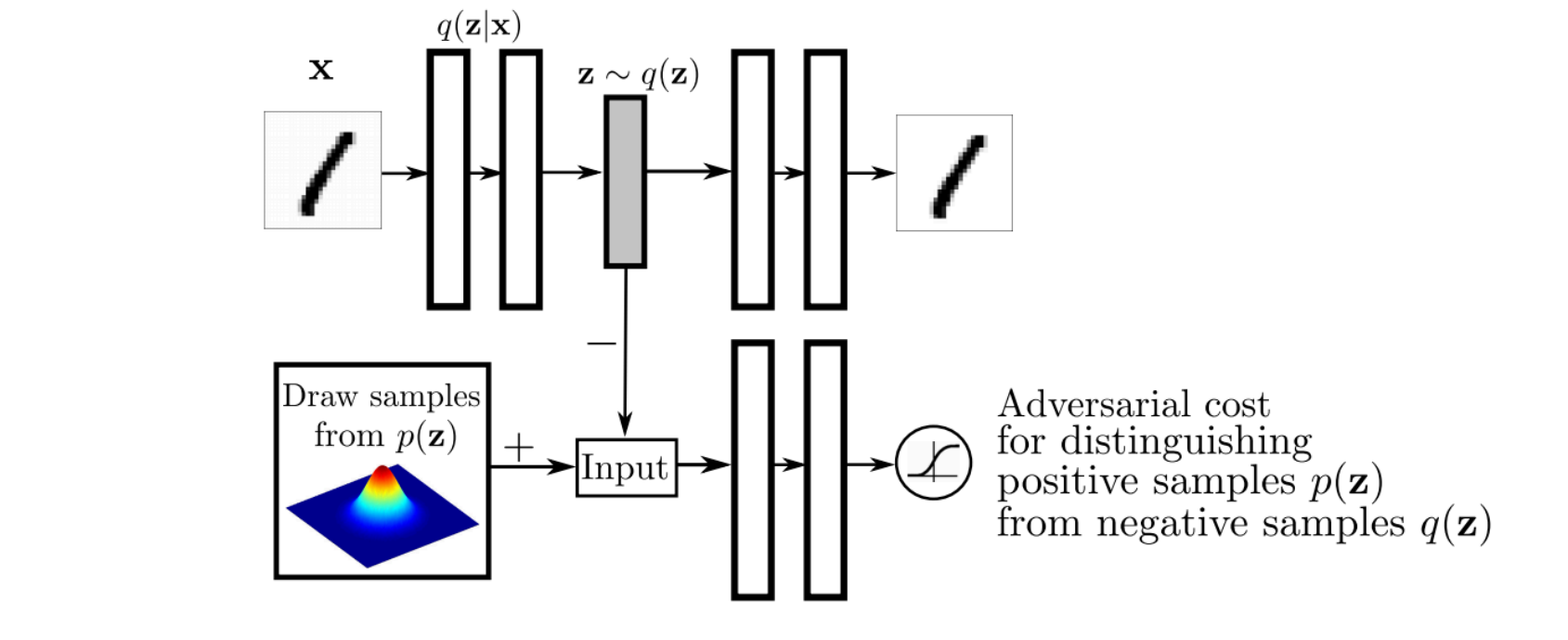
SAGAN

- Трудно оценивать качество картинок.
- Inception score (Salimans et al., 2016; вообще важная статья о GAN'ax):
  - берём порождённые картинки $\mathbf{x}$;
  - применяем стандартный классификатор Inception;
  - хотим, чтобы у $p(y \mid \mathbf{x})$ была маленькая энтропия, но разнообразие чтобы было, т.е. чтобы у $p(y) = \int p(y \mid \mathbf{x} = G(\mathbf{z}))\mathrm{d}\mathbf{z}$ была большая энтропия;
  - т.е. получаем метрику $\exp(\mathbb{E}_{\mathbf{x}}\mathrm{KL}(p(y \mid \mathbf{x}) \| p(y)))$;
  - для обучения это трудно приспособить, а вот для оценки качества хорошо.

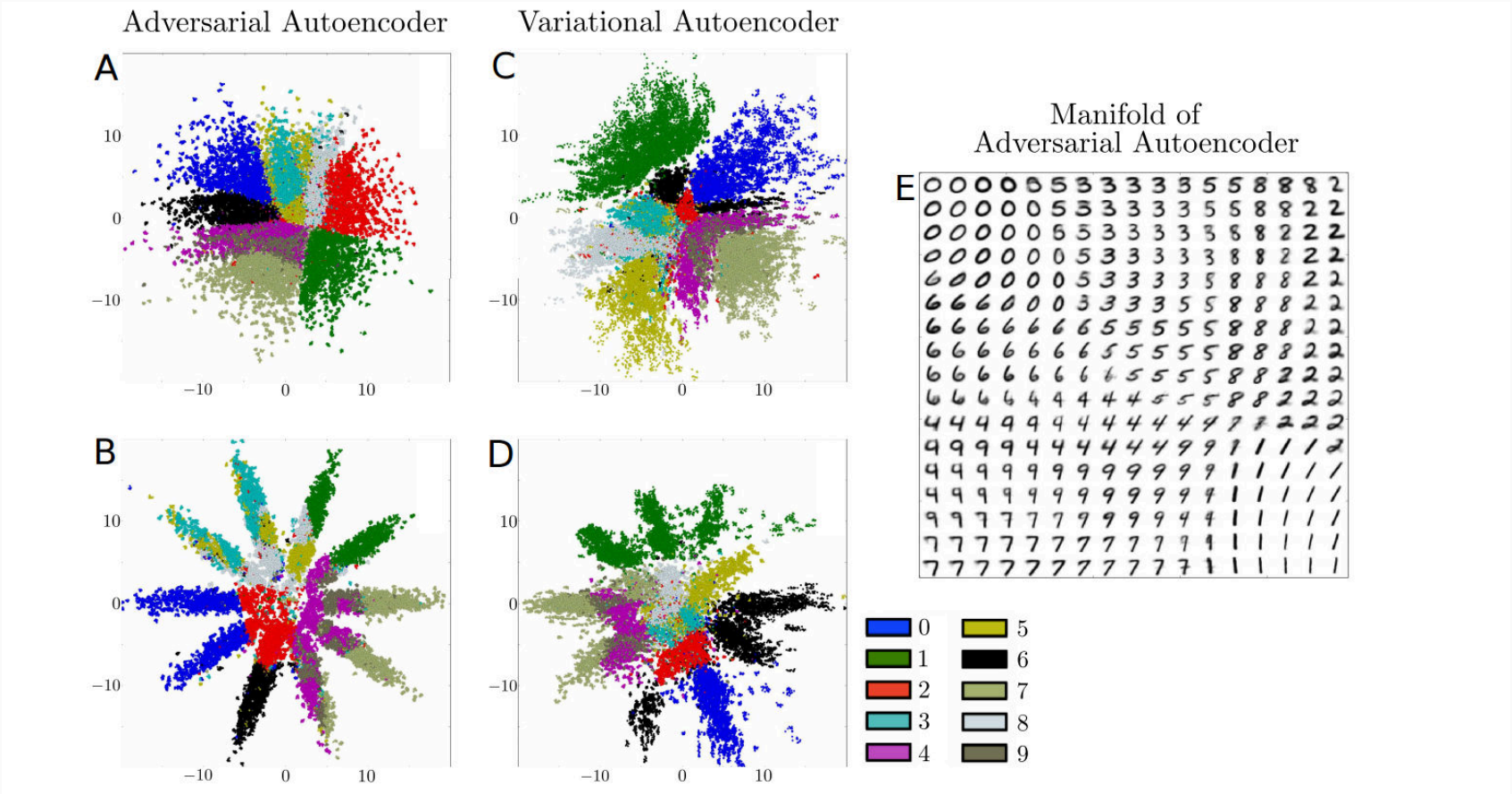
Архитектуры, основанные на GAN

AAE

- Соперничающие автокодировщики (adversarial autoencoders; Makhzani et al., 2015): дискриминатор приводит распределение признаков (на скрытом слое) к заданному $p(\mathbf{z})$.

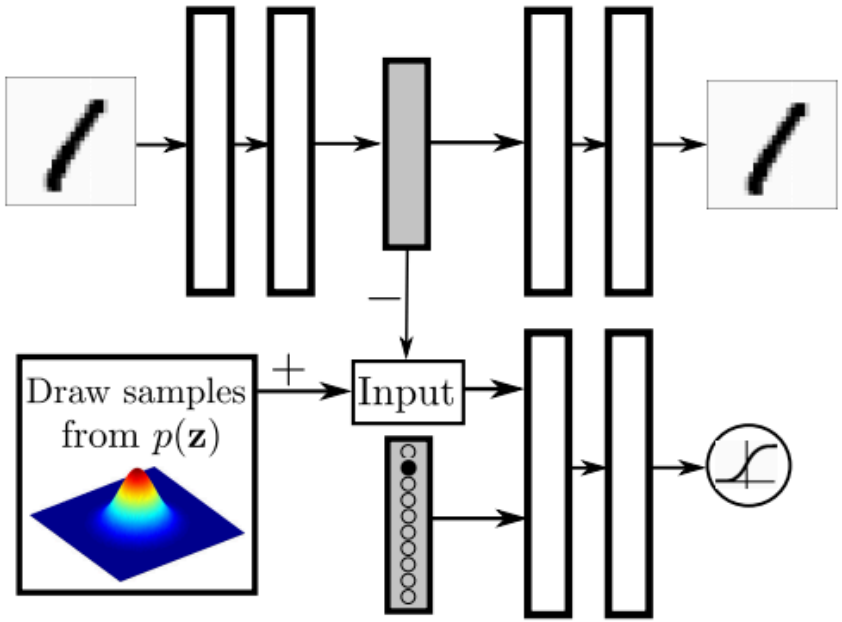


AAE

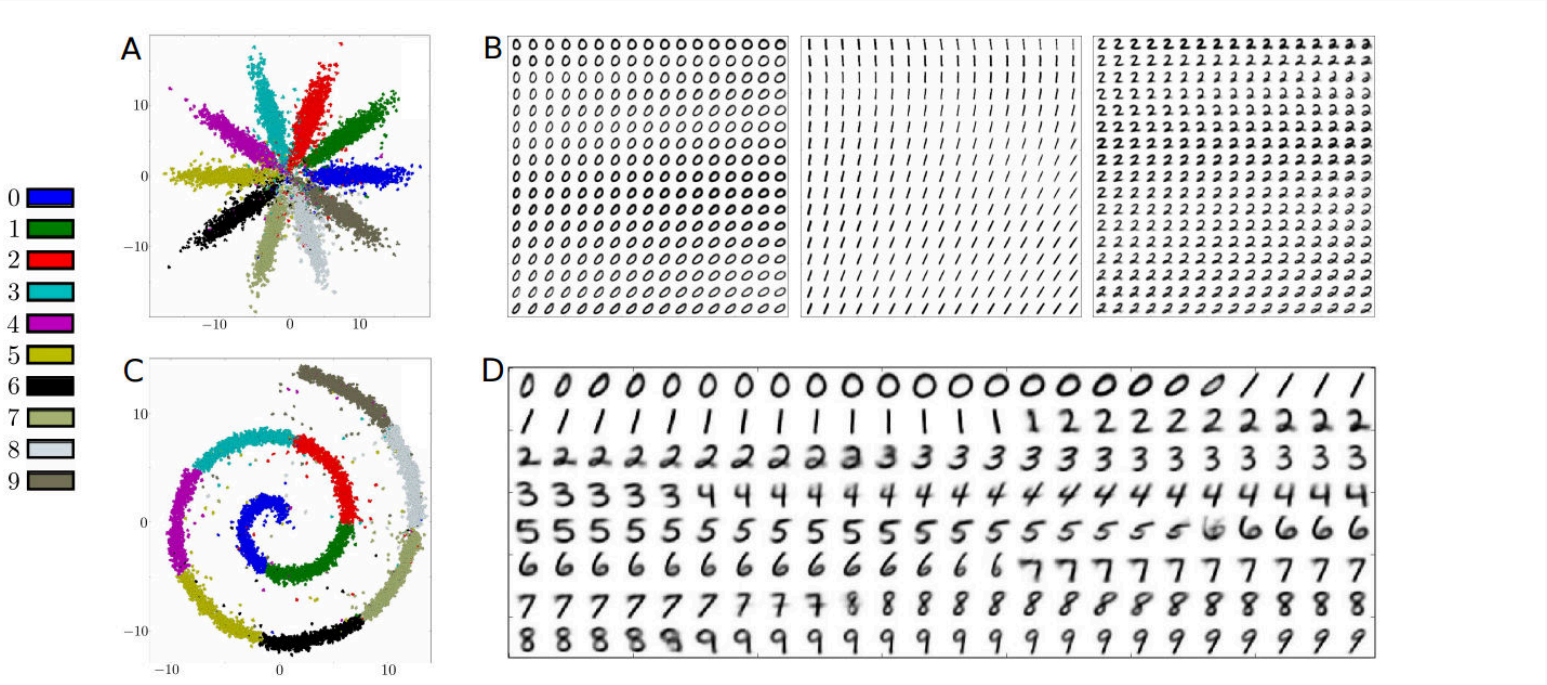

Adversarial Autoencoder
Variational Autoencoder
A
B
C
D
10
0
−10
−10
0
10
Manifold of
Adversarial Autoencoder
E
0
1
2
3
4
5
6
7
8
9

AAE

- Можно сделать распределения условными и получить semi-supervised learning:

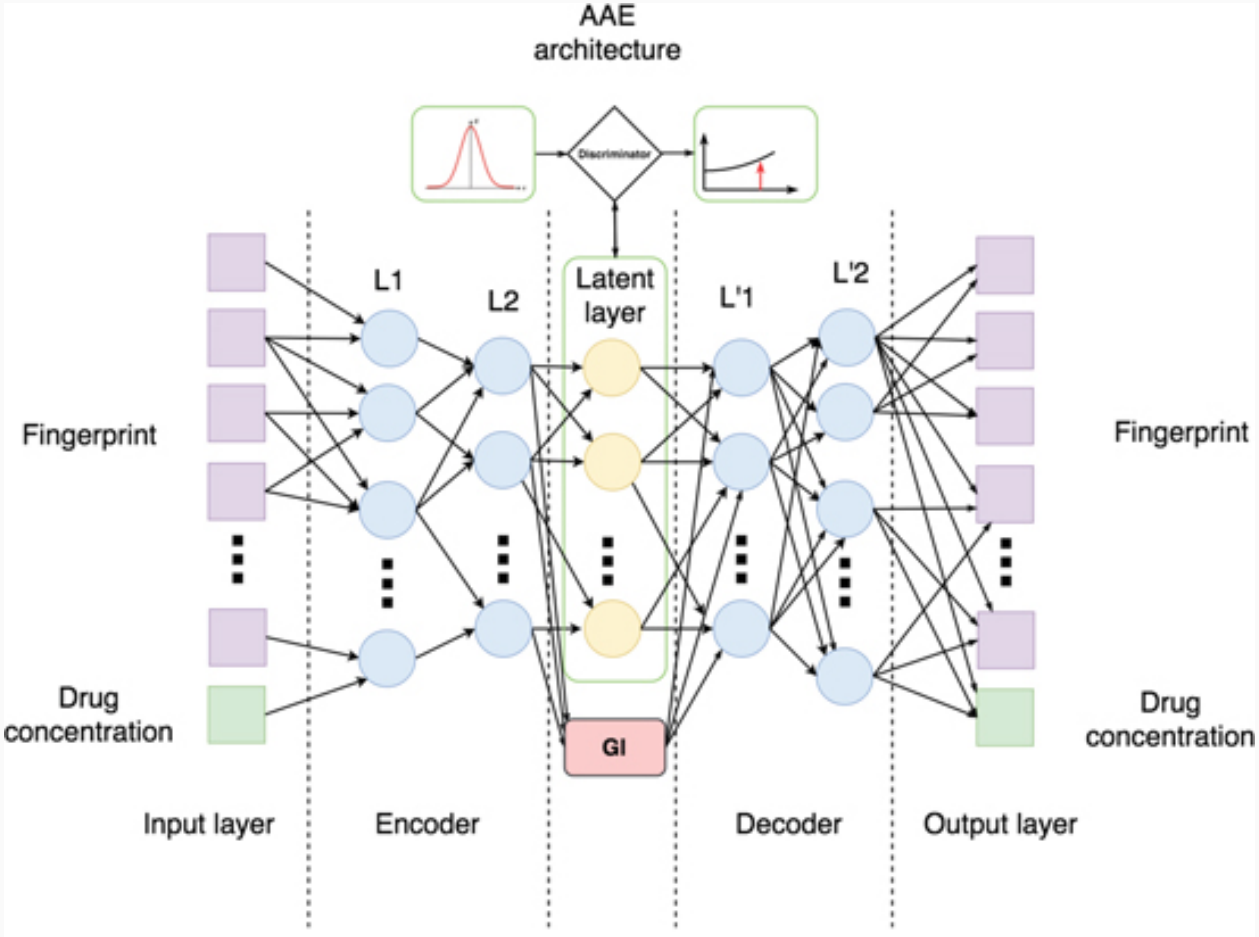


AAE



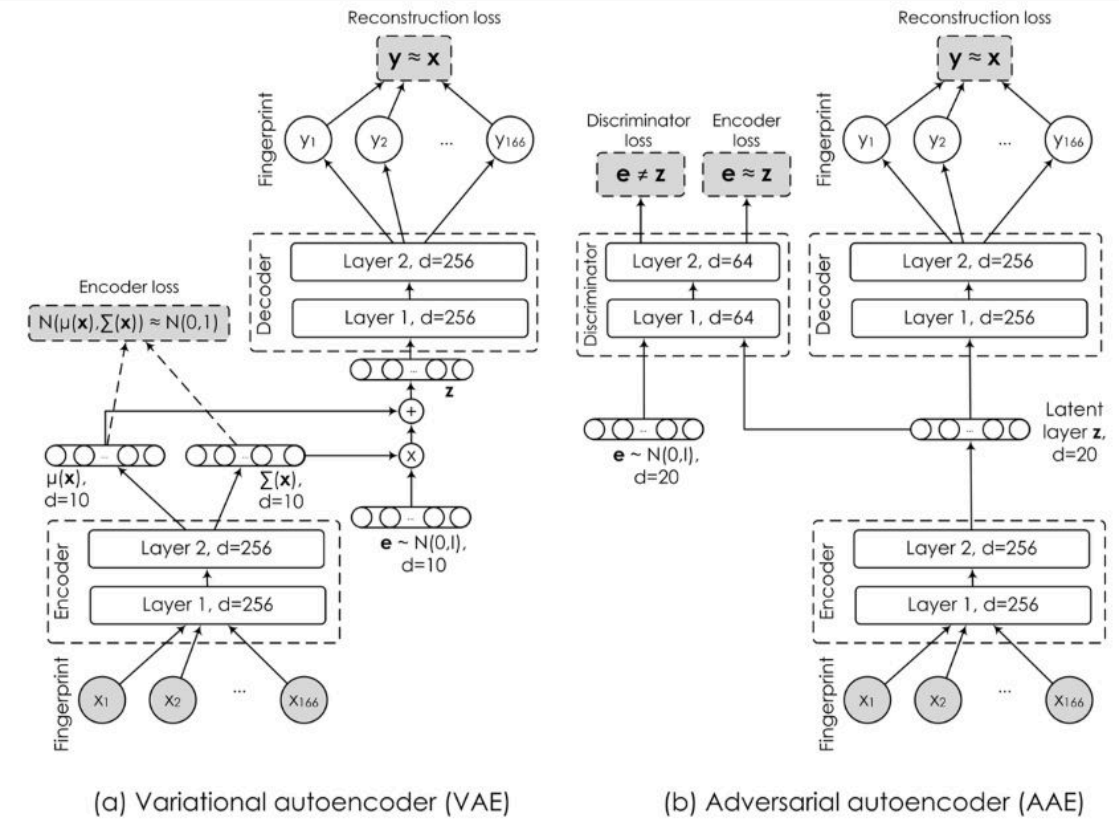
AAE

- (Kadurin et al., 2017) – AAE для порождения молекул в онкологии:



AAE

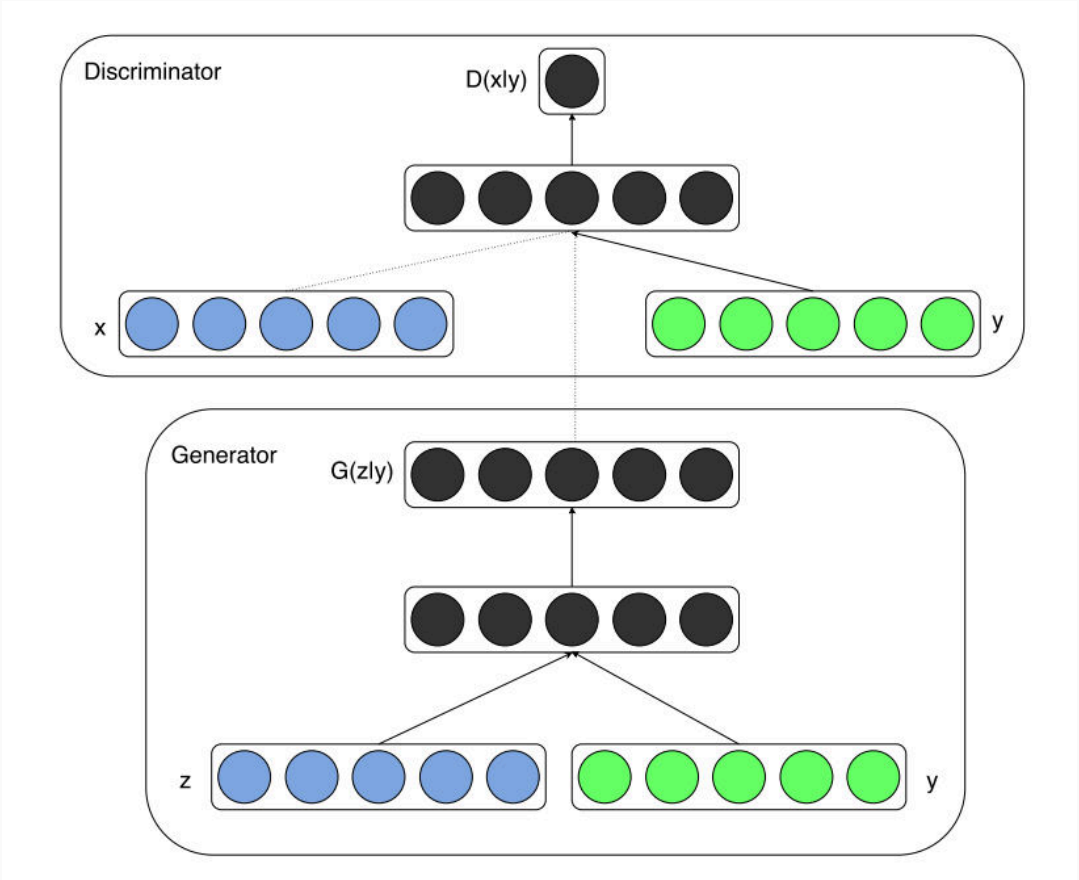
- (Kadurin et al., 2018) – druGAN для порождения молекул; сравнили с VAE:



- (Polikovsky et al., 2018): MOSES – платформа для сравнения порождающих моделей для молекул

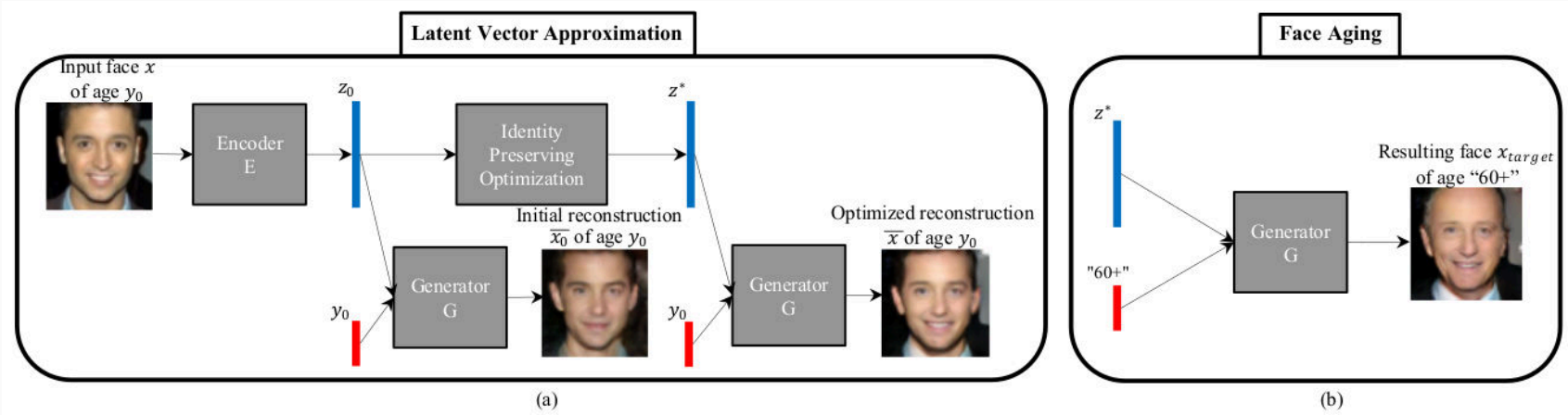
Условные GAN'ы

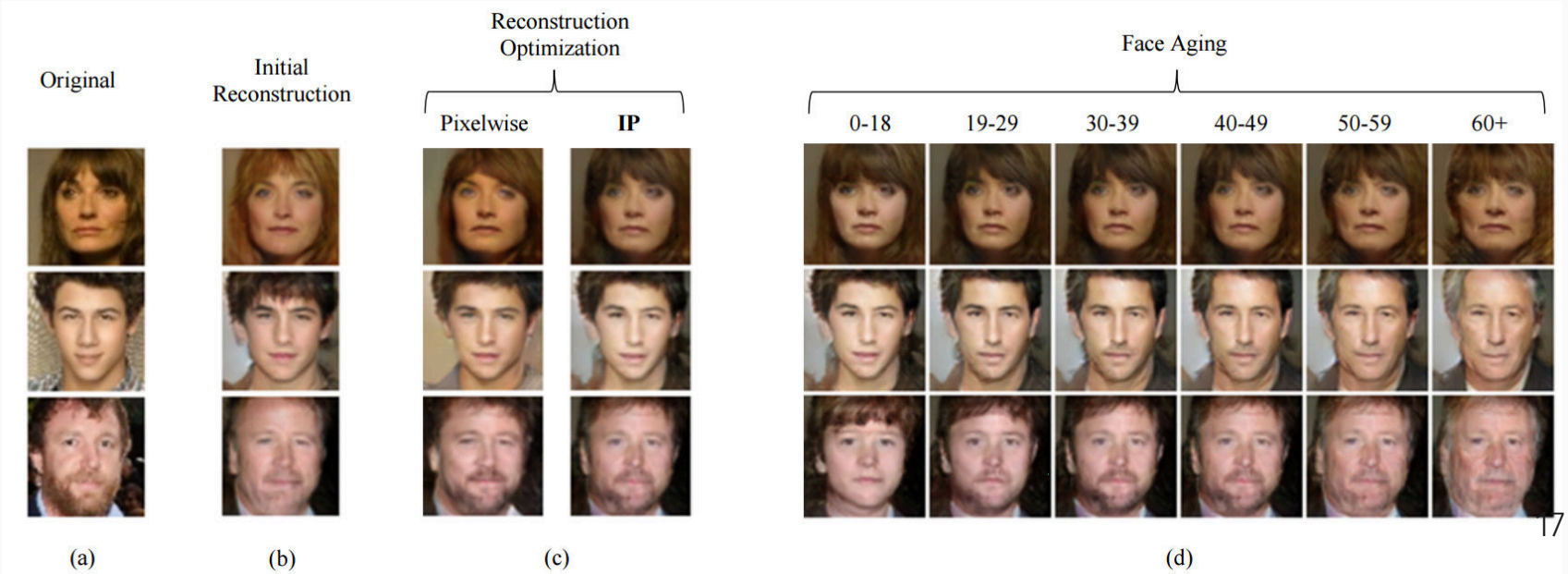
- Условные GAN’ы (conditional GANs; Mirza and Osindero, 2014): G получает случайный вектор и вектор входа, вход – это условие.



Условные GAN'ы

- (Antipov et al., 2017) – состарим лицо условным GAN’ом:





Условные GAN'ы

- (Ledig et al., 2017) – ESRGAN для superresolution (но об этом можно долго разговаривать)

Stacked GANs

- Часто полезно сделать несколько GAN'ов подряд.
- (Huang et al., 2017): Stacked Generative Adversarial Networks
- G_i последовательно обучает более низкоуровневые представления, обращая глубокую сеть из E_i.
- Новые loss functions:
  - conditional loss – мы обучаем $\hat{\mathbf{h}}_i$ при условии $\mathbf{h}_{i+1}$, и чтобы генератор не игнорировал условие, надо, чтобы восстанавливались $\mathbf{h}_{i+1}$ через соответствующий encoder:

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

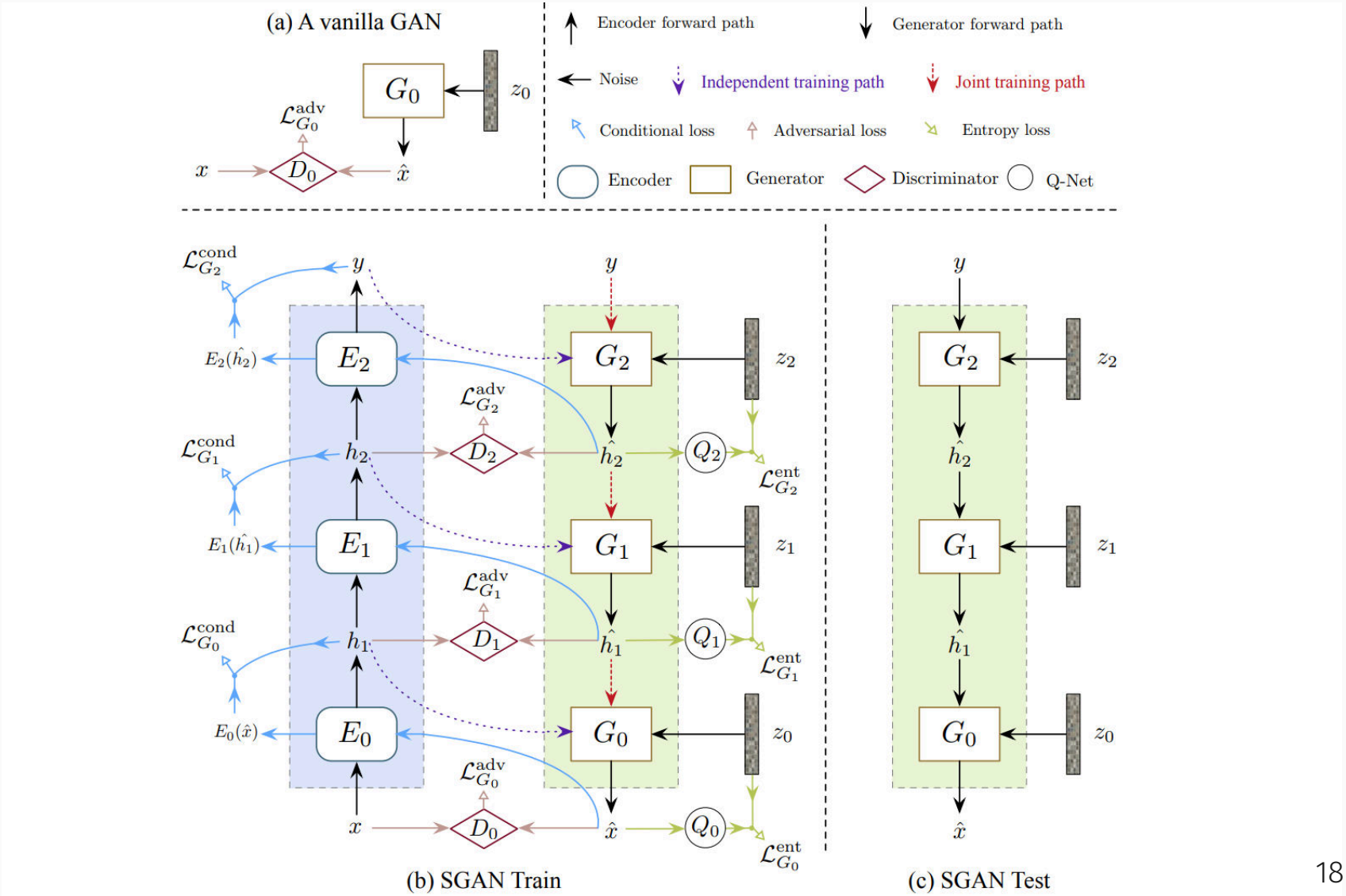
  - entropy loss – но и надо, чтобы G_i не игнорировал $\mathbf{z}$, надо добавить diversity; для этого максимизируем энтропию:

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

    где Q_i – параметризованное нейросетью приближение для апостериорного распределения $p_i(\mathbf{z}_i \mid \hat{\mathbf{h}}_i)$ (вариационная оценка).

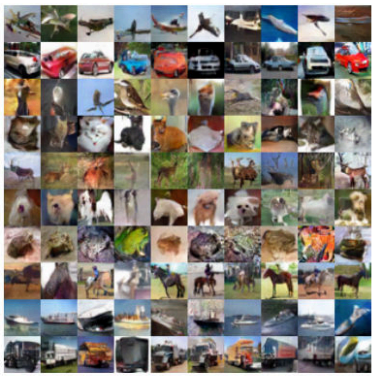
Stacked GANs

- SGAN:

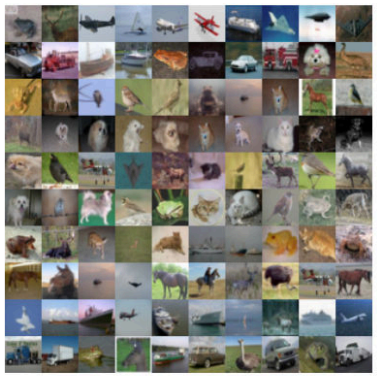


Stacked GANs

- Получается лучше, чем у предшественников (хотя ещё лучше progressive growing):

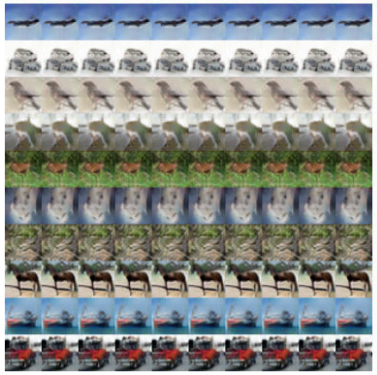


(a) SGAN samples (conditioned on labels)



(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

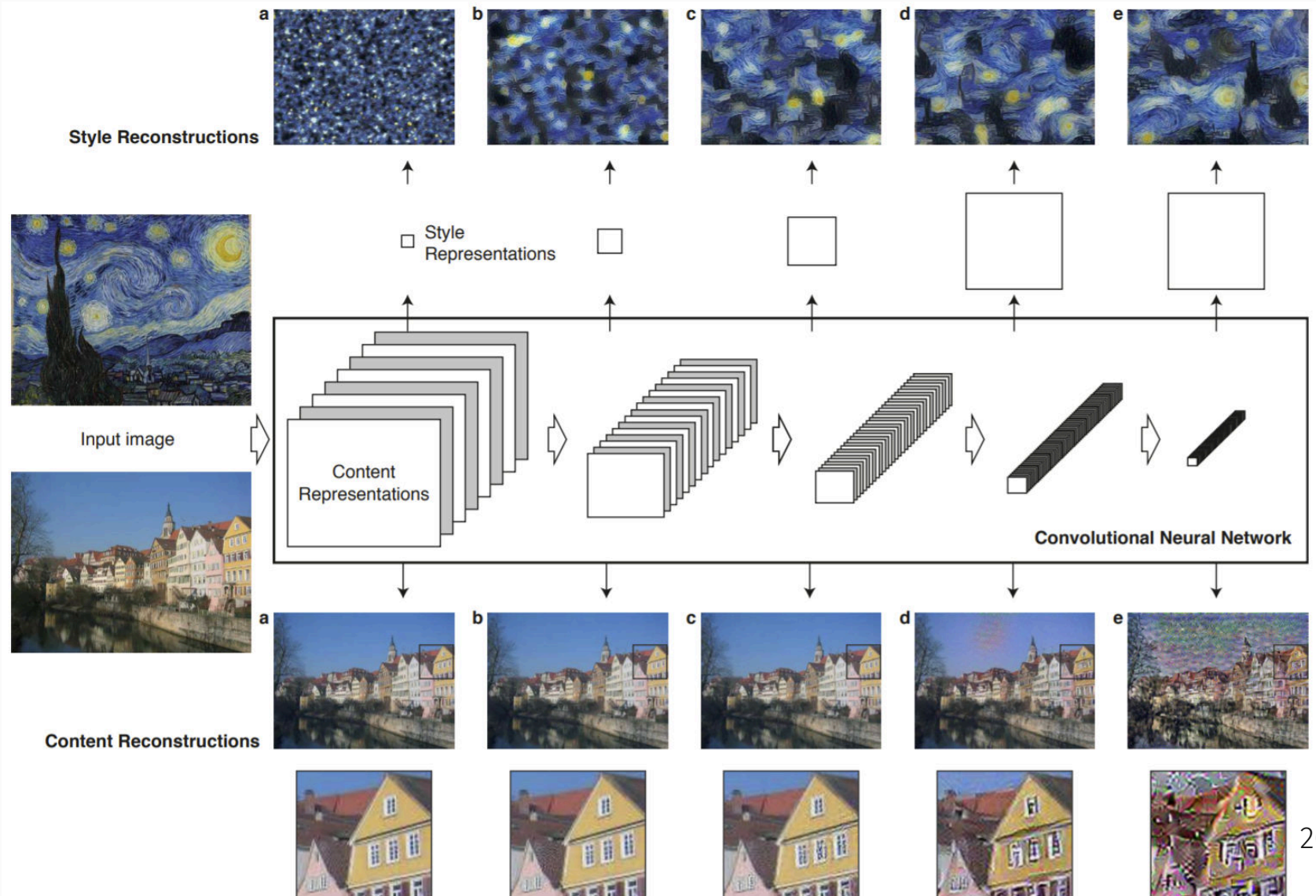
† Trained with labels.

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

Case study: перенос стиля

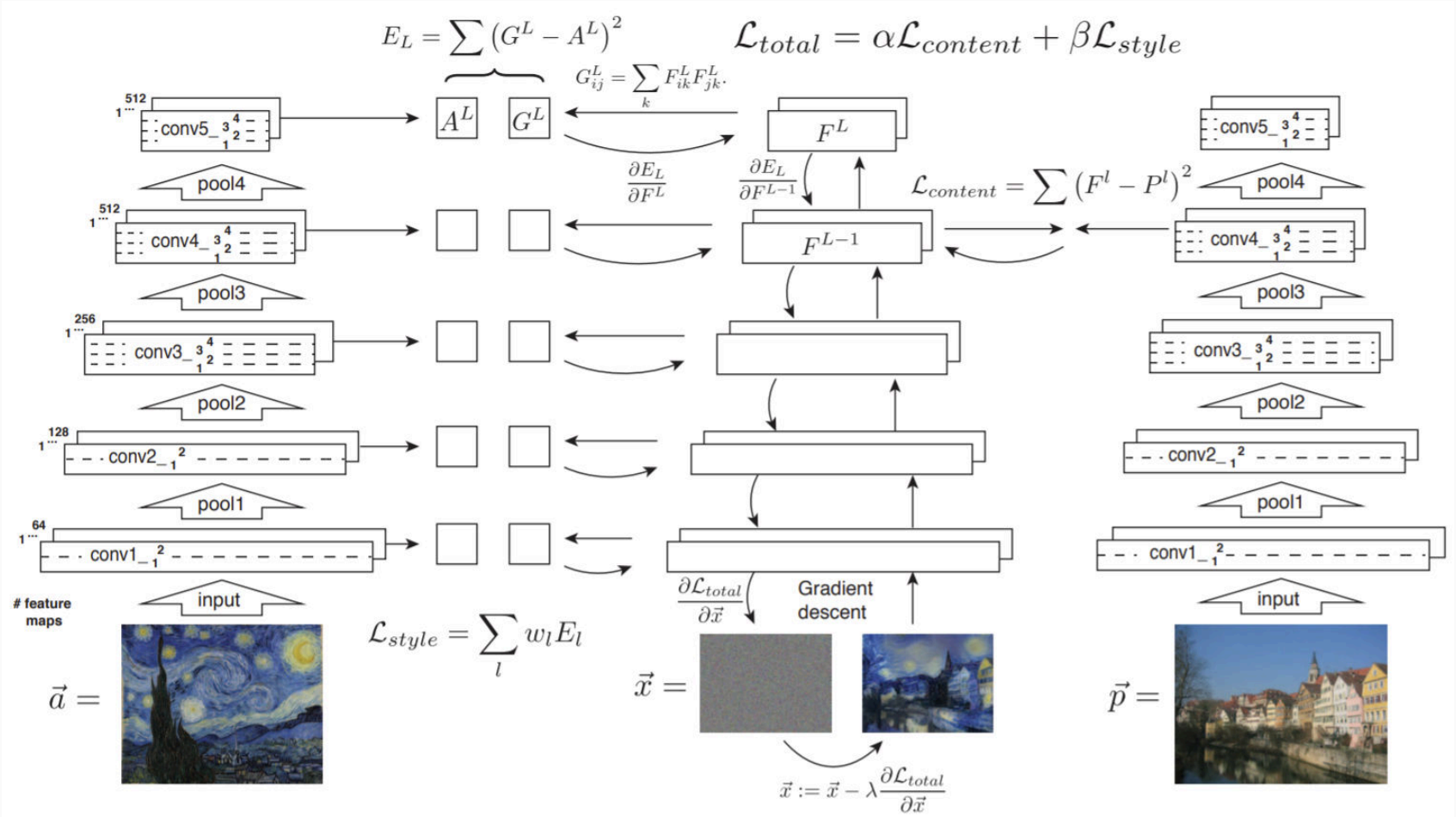
Перенос стиля до GAN'ов

- (Gatys et al., 2015): перенос стиля свёрточными сетями



Перенос стиля до GAN'ов

- (Gatys et al., 2015): перенос стиля свёрточными сетями



Перенос стиля до GAN'ов

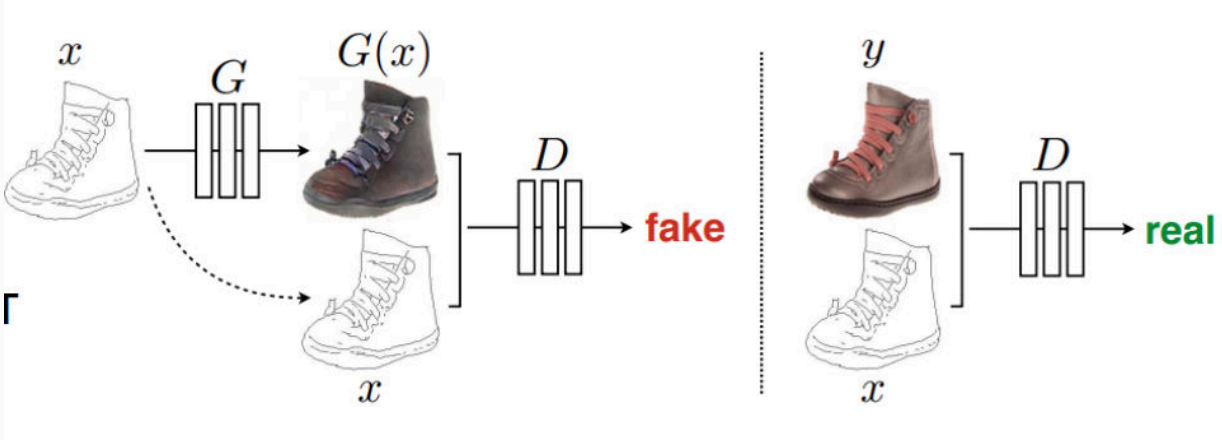
- (Gatys et al., 2015): перенос стиля свёрточными сетями

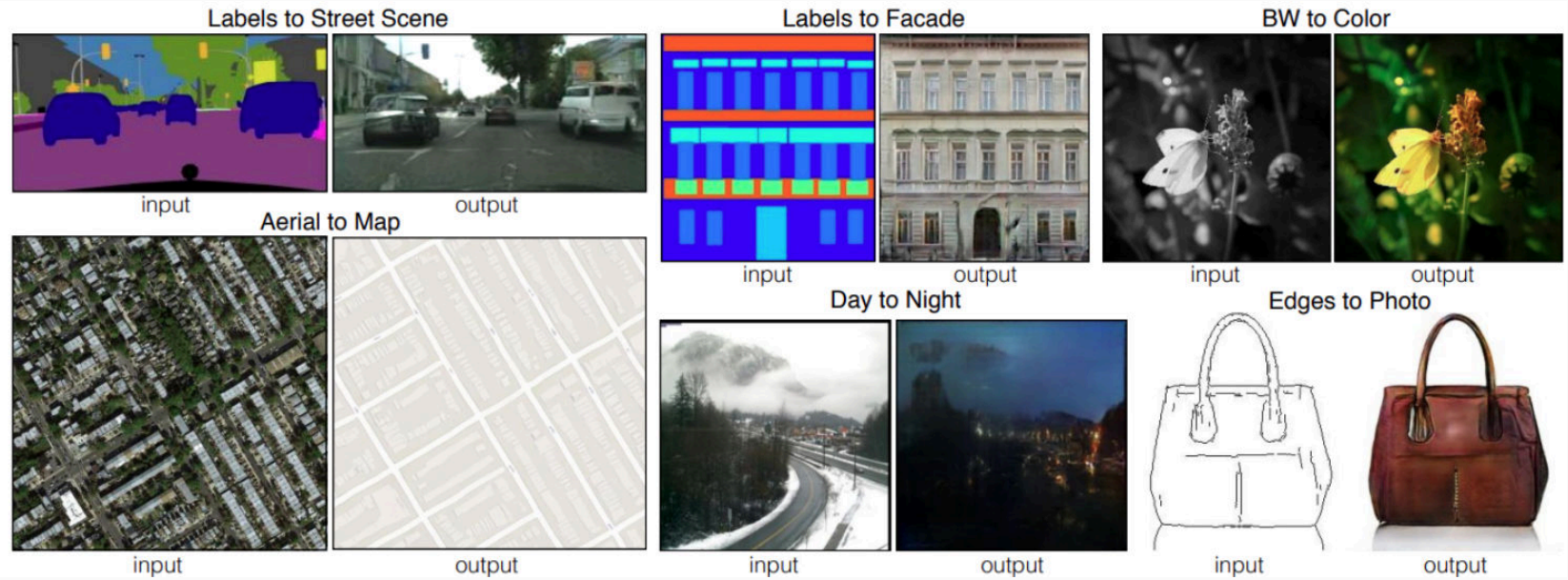
Перенос стиля до GAN'ов

- (Gatys et al., 2015): перенос стиля свёрточными сетями

Перенос стиля с GAN'ами

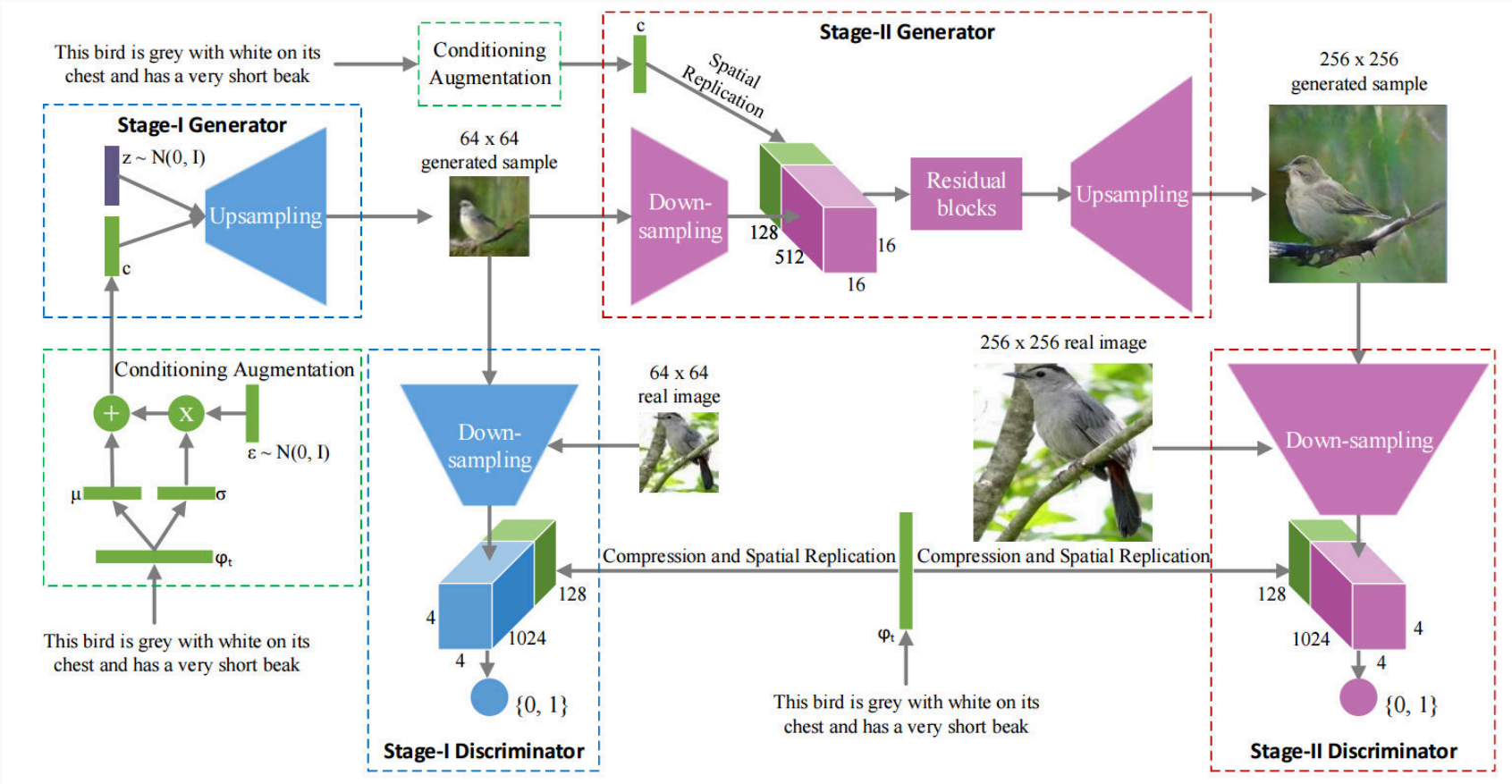
- pix2pix (Isola et al., 2017): отлично работает с paired dataset





Перенос стиля с GAN'ами

- Стиль не обязан быть изображением!
- StackGAN (Zhang et al., 2016) – по тексту породим картинку:

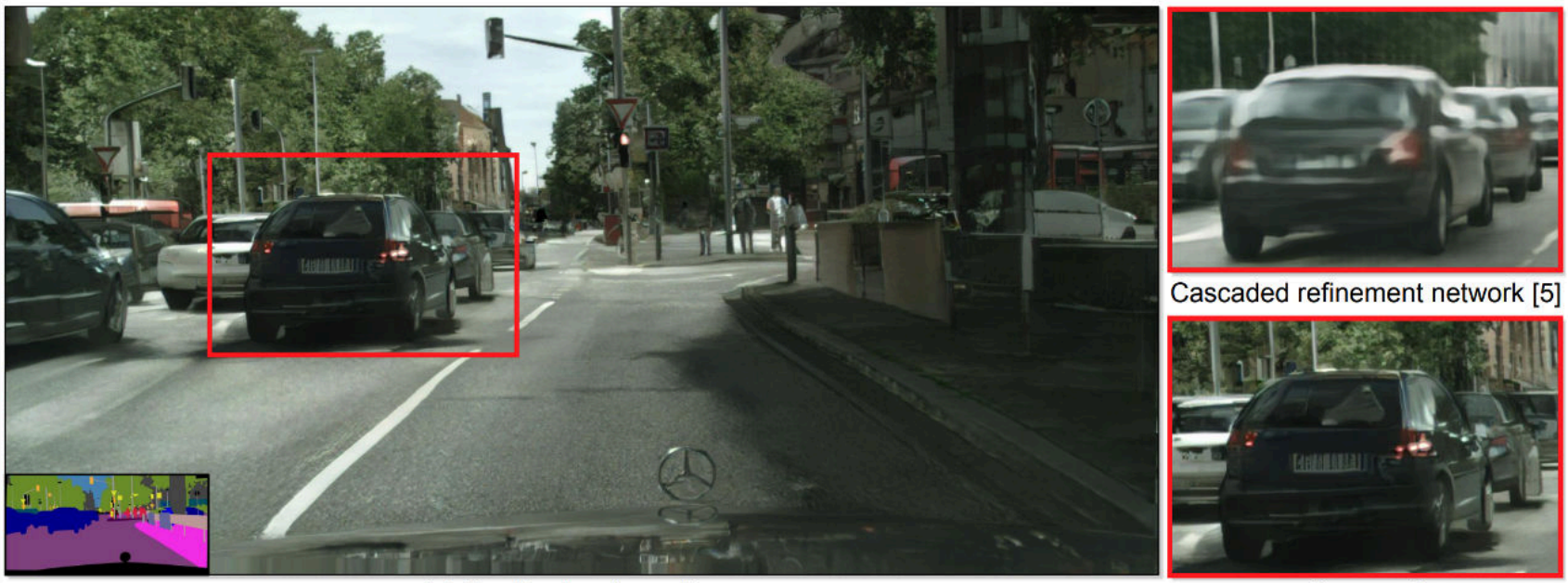


Перенос стиля с GAN'ами

- Вот результаты реального порождения (best of 16):

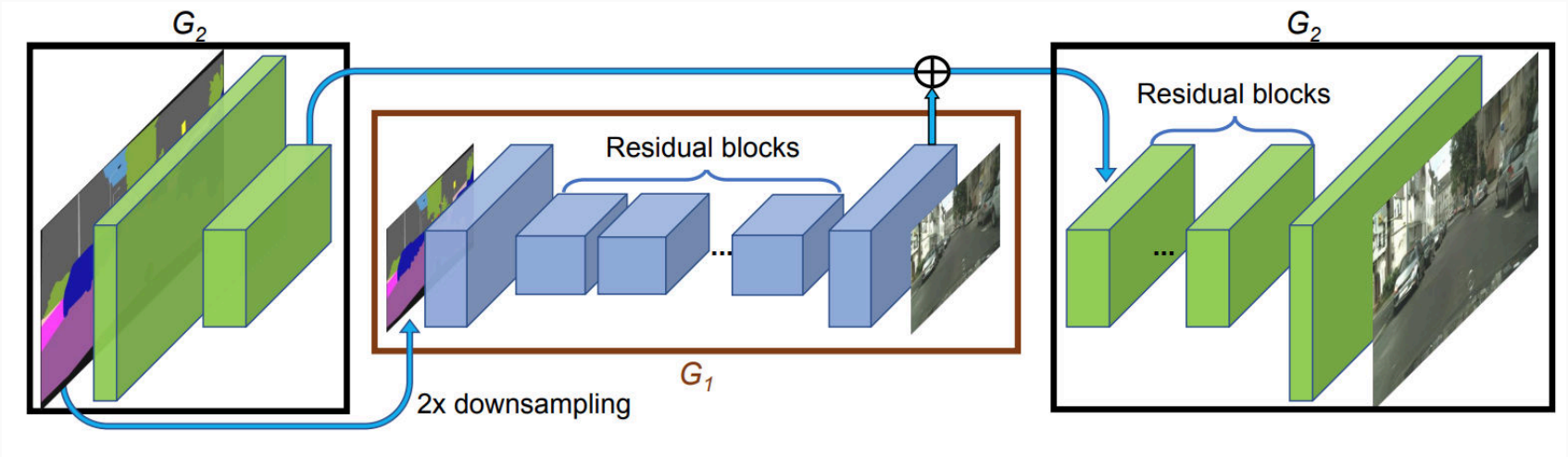
Перенос стиля с GAN'ами

- pix2pixHD (Wang et al., 2017) – то же самое в высоком разрешении



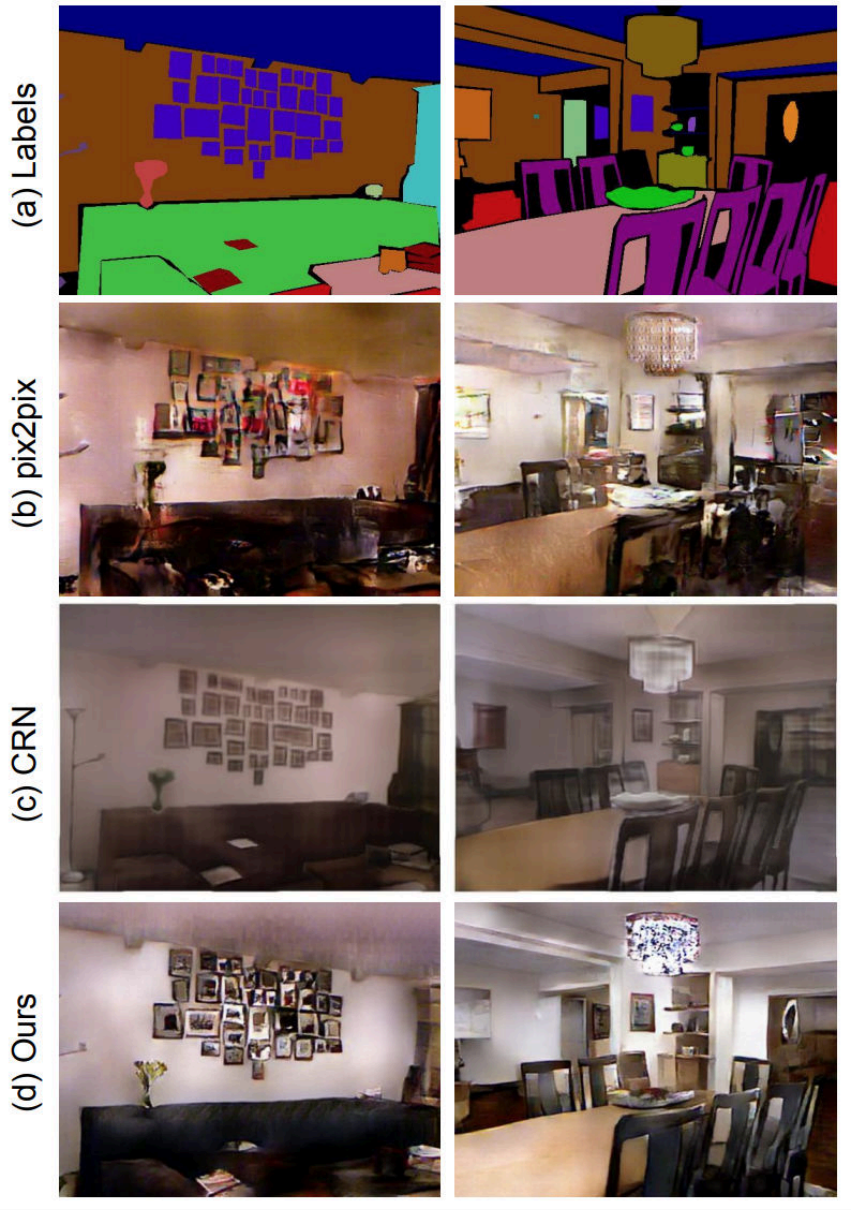
Перенос стиля с GAN'ами

- Генератор теперь из двух частей, вторая улучшает результат первой



Перенос стиля с GAN'ами

- Ещё кое-какие трюки, в общем, лучше получается:



Перенос стиля с GAN'ами

- А параллельно с этим Huang и Belongie придумали AdaIN (adaptive instance normalization):
  - batch normalization использует статистики по мини-батчу:

$$\mathrm{BN}(x) = \gamma\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \beta$$

  - instance normalization $\mathrm{IN}(x)$ – это то же самое, но статистики по каждому каналу и каждой картинке по отдельности
  - conditional instance normalization – это когда γ и β обучаются для каждого стиля:

$$\mathrm{CIN}(x; s) = \gamma_s\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \beta_s$$

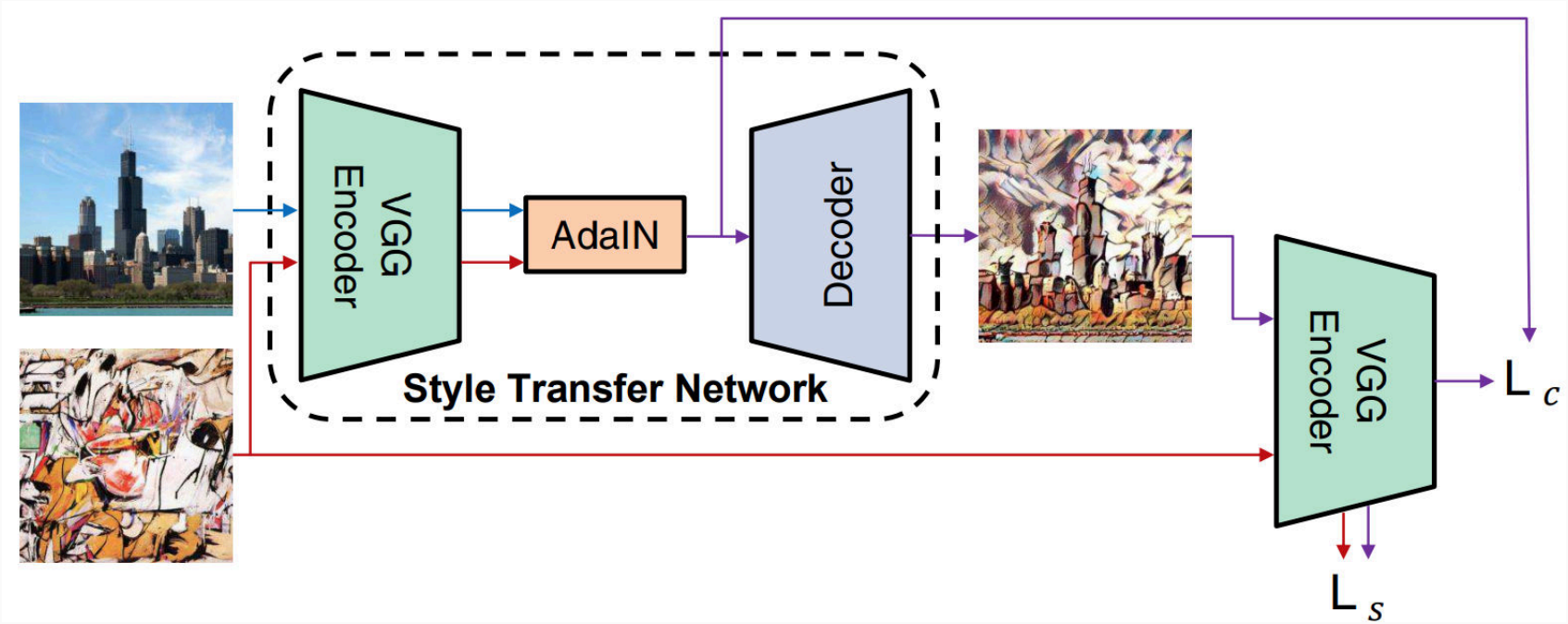
  - AdaIN – это то же самое, но мы просто берём параметры от другой картинки, которая задаёт стиль:

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

- Чем-то похоже на самый ранний artistic style transfer (Gatys et

Перенос стиля с GAN'ами

- Теперь сама сеть супер-простая – AdaIN действует в feature space какого-нибудь автокодировщика:

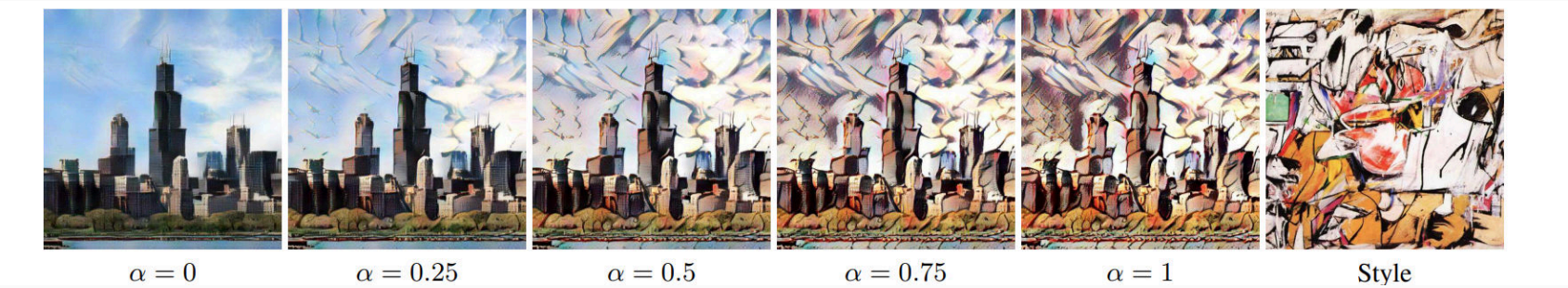


Перенос стиля с GAN'ами

- И получается круто:

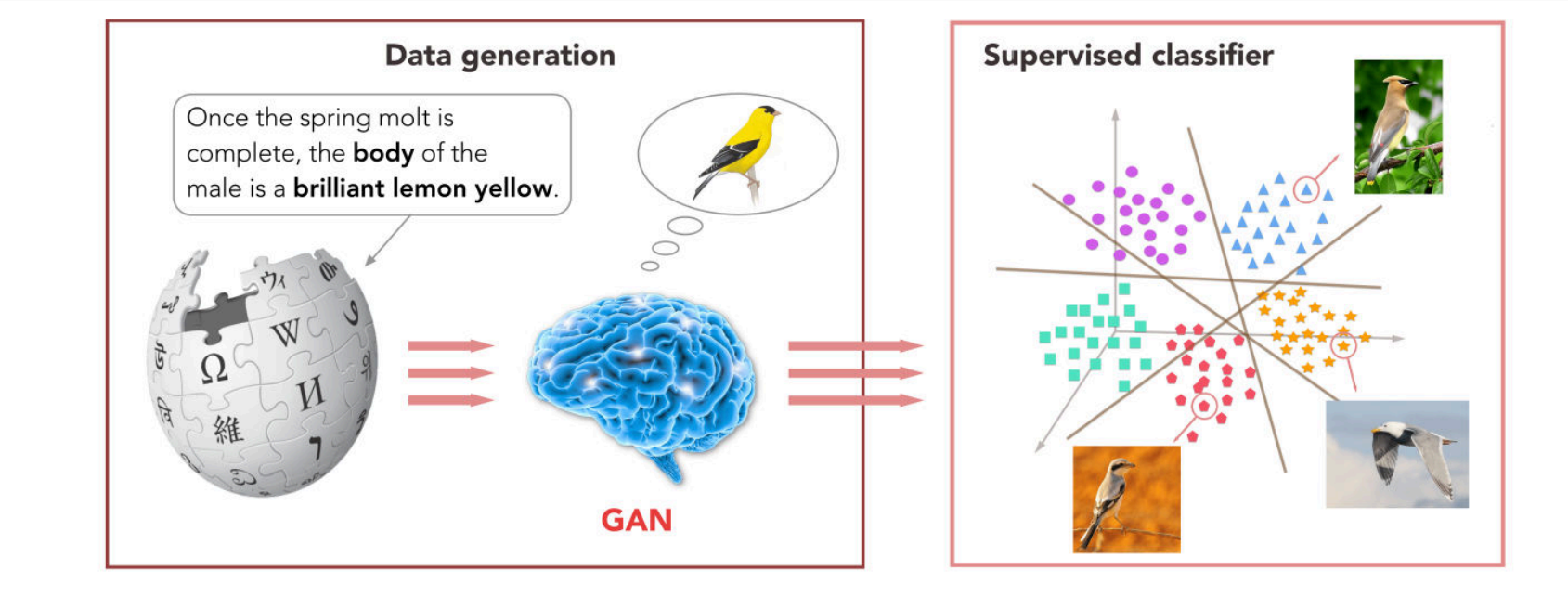
Перенос стиля с GAN'ами

- А ещё можно интерполировать и задавать «силу» стиля:



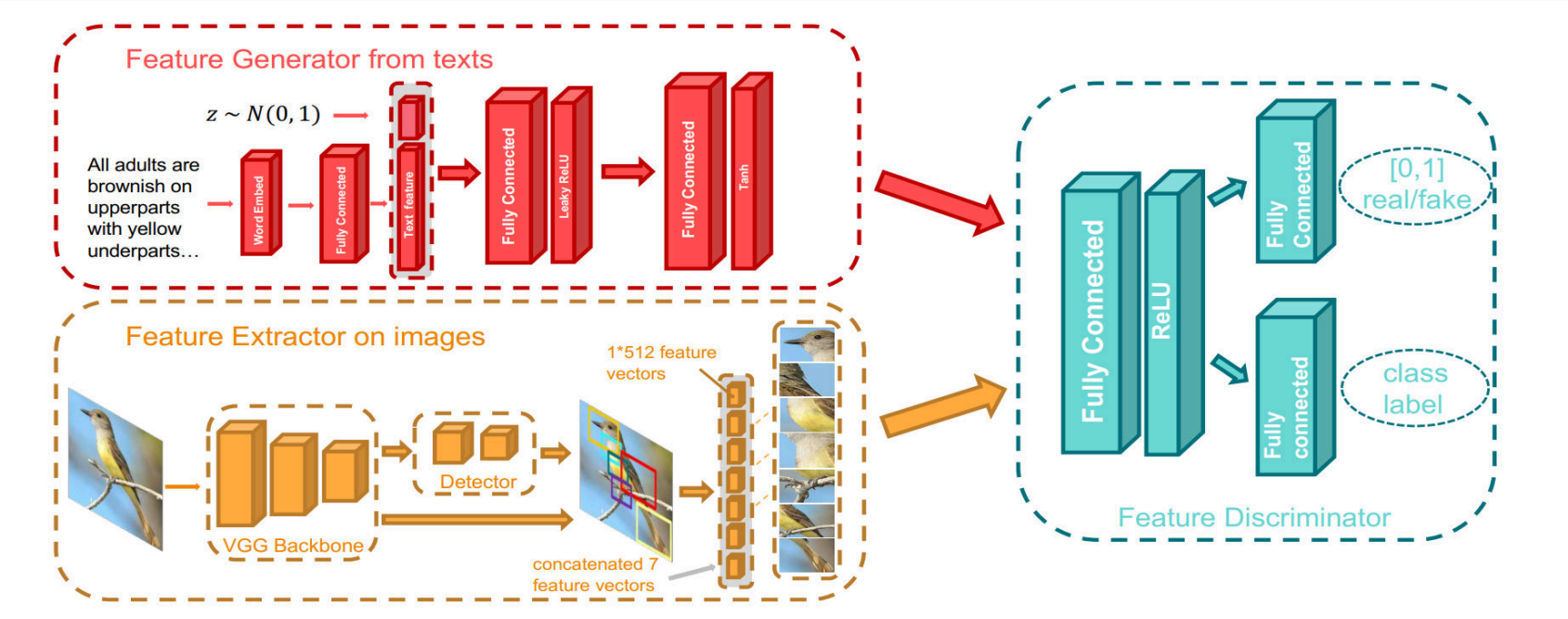
Zero-Shot Learning

- (Zhu et al., CVPR 2018): A Generative Adversarial Approach for Zero-Shot Learning from Noisy Texts
- Для 2018 это были вообще безумные вещи – можно обучать классификатор картинок на текстах из википедии:



Zero-Shot Learning

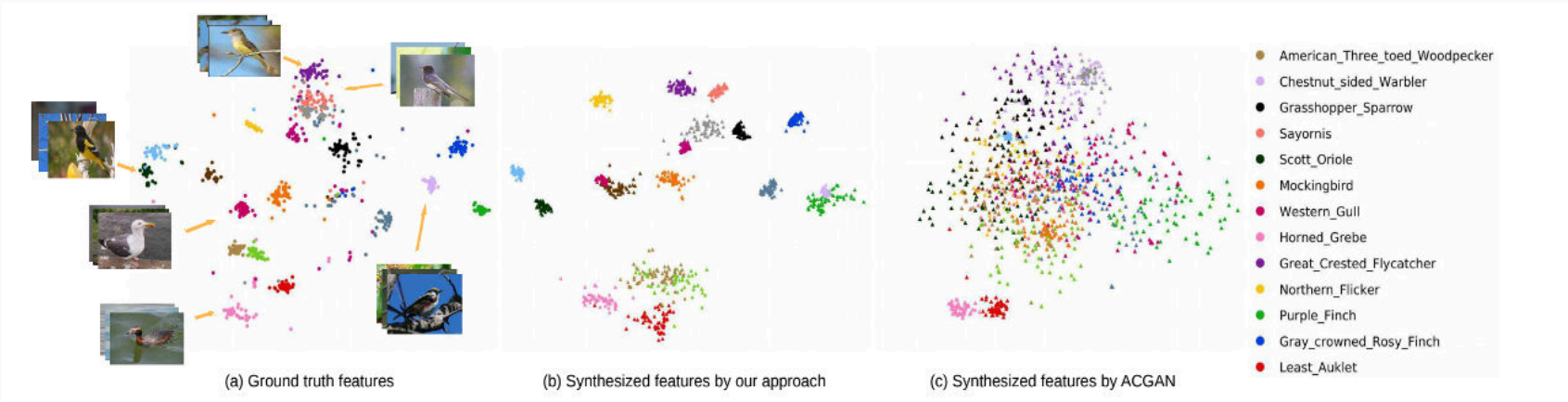
- Логичная архитектура:



- Тут важно, что для классификатора не надо порождать картинки, можно просто признаки порождать.

Zero-Shot Learning

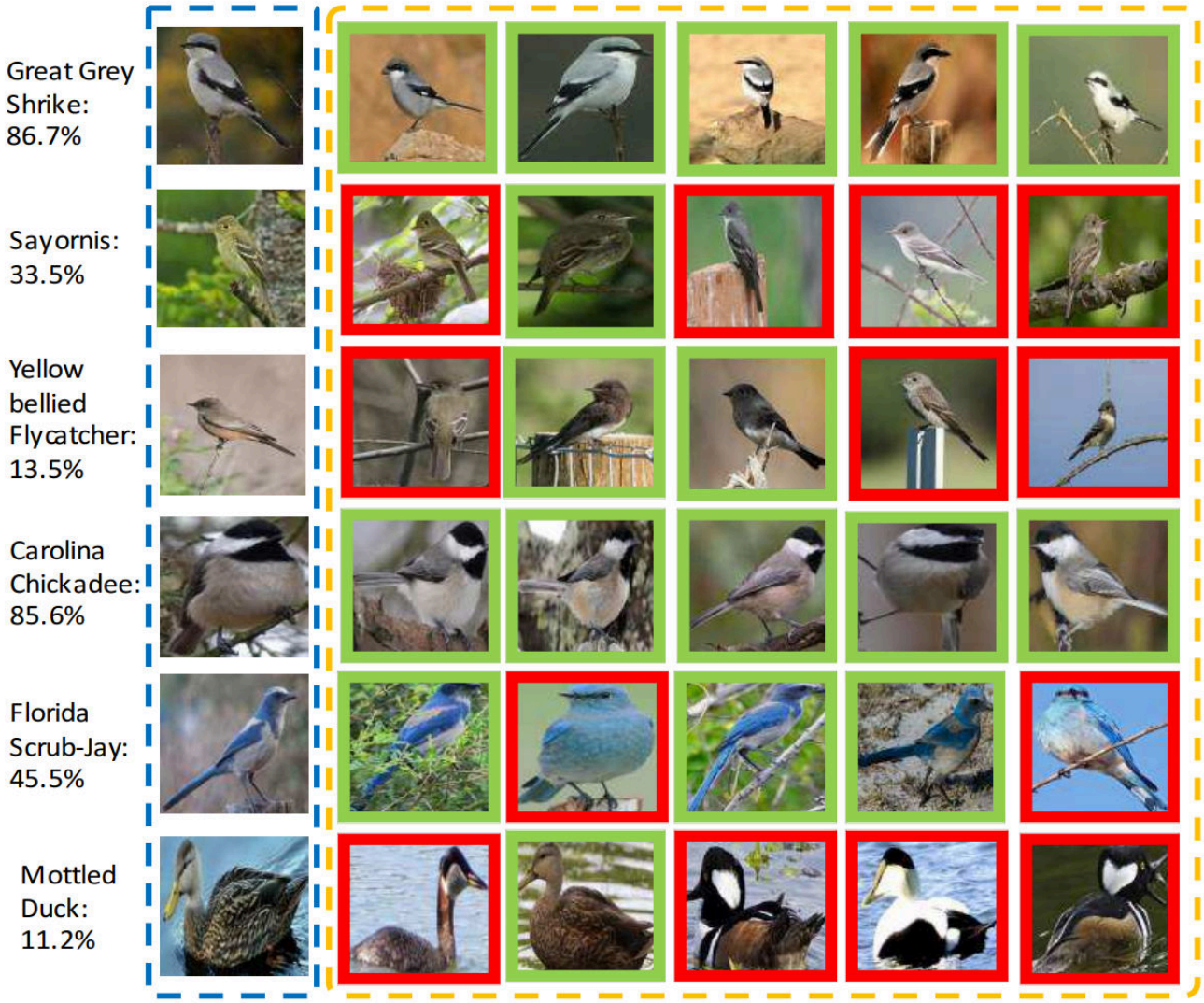
- И ведь работает:



(a) Ground truth features (b) Synthesized features by our approach (c) Synthesized features by ACGAN

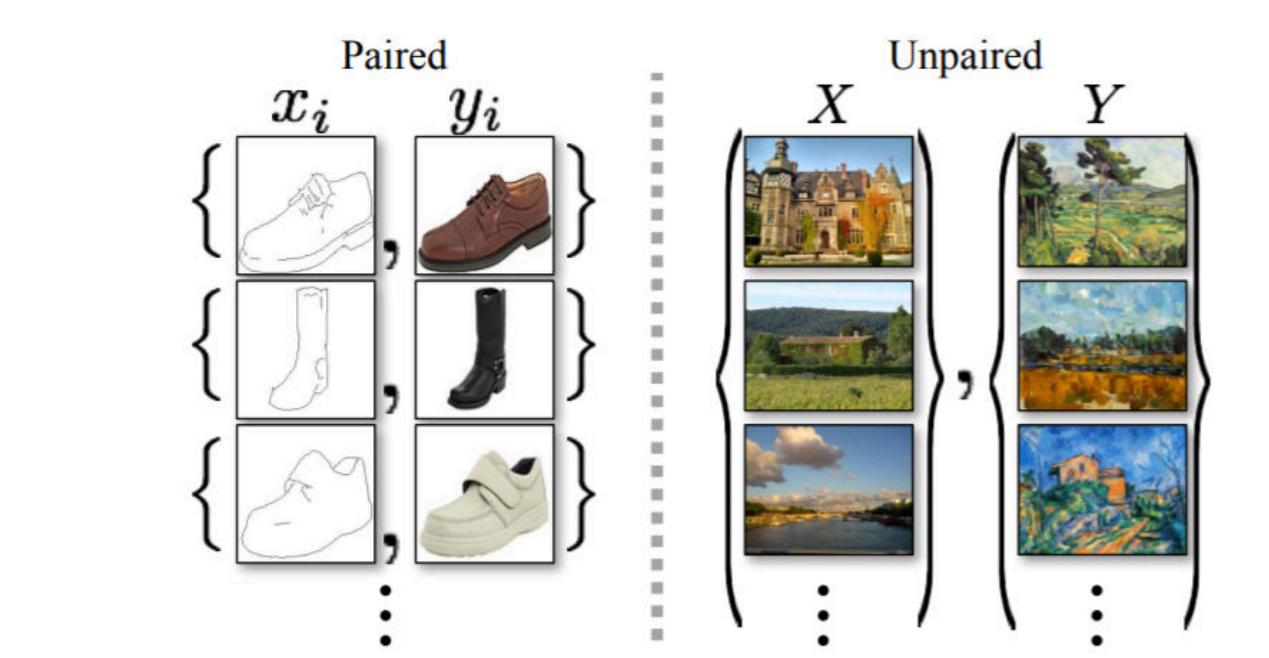
Zero-Shot Learning

- Можно делать zero-shot retrieval: искать картинки птичек, которых модель никогда не видела



CycleGAN

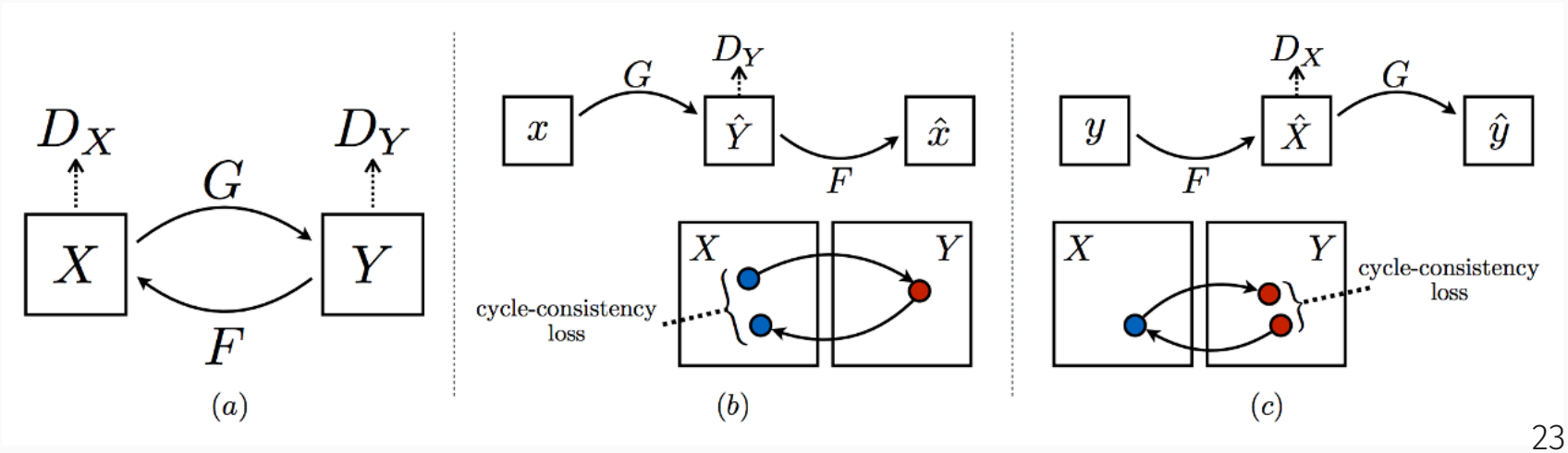
- CycleGAN (Zhu et al., 2017)
  - что делать, если данные unpaired?
  - например, для style transfer:



CycleGAN

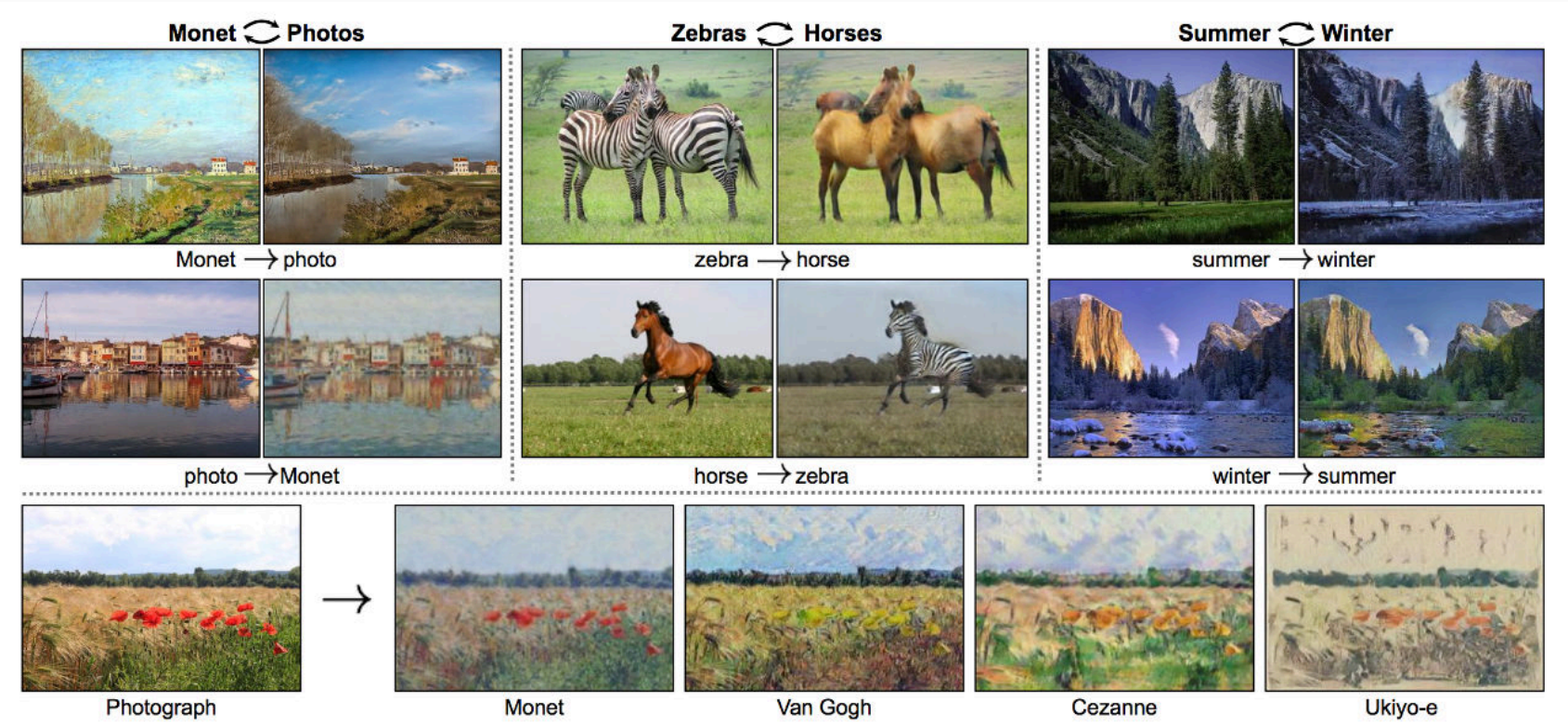
- CycleGAN (Zhu et al., 2017)
- Основная идея в том, что после двойного цикла style transfer должно опять получиться то же самое, $G(F(\mathbf{x})) = \mathbf{x}$
- Общая функция потерь складывается из двух GAN'ов для G и F и cycle loss:

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) + \lambda \mathcal{L}_{\text{cyc}}(G, F)$$



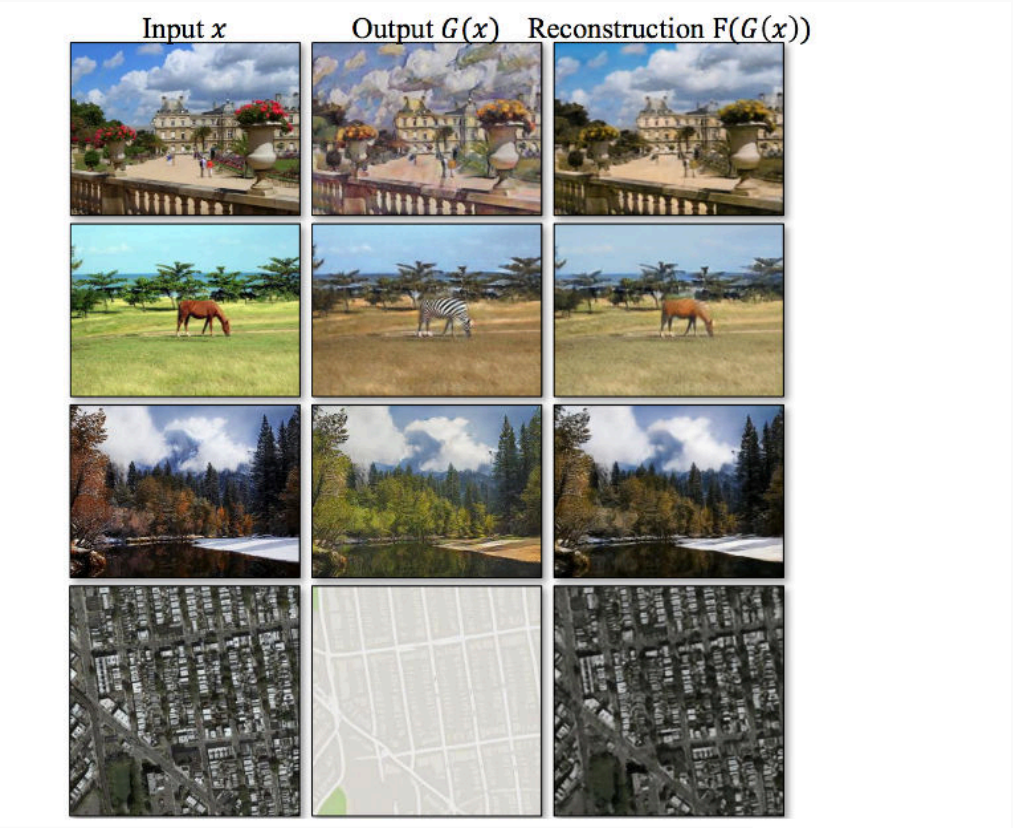
CycleGAN

- CycleGAN (Zhu et al., 2017)
- И получается очень хороший style transfer без всяких выровненных данных



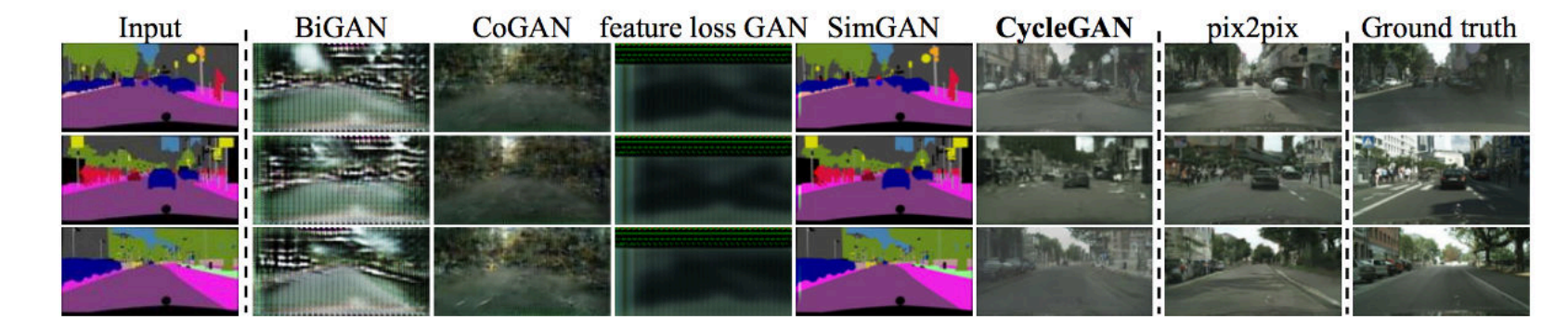
CycleGAN

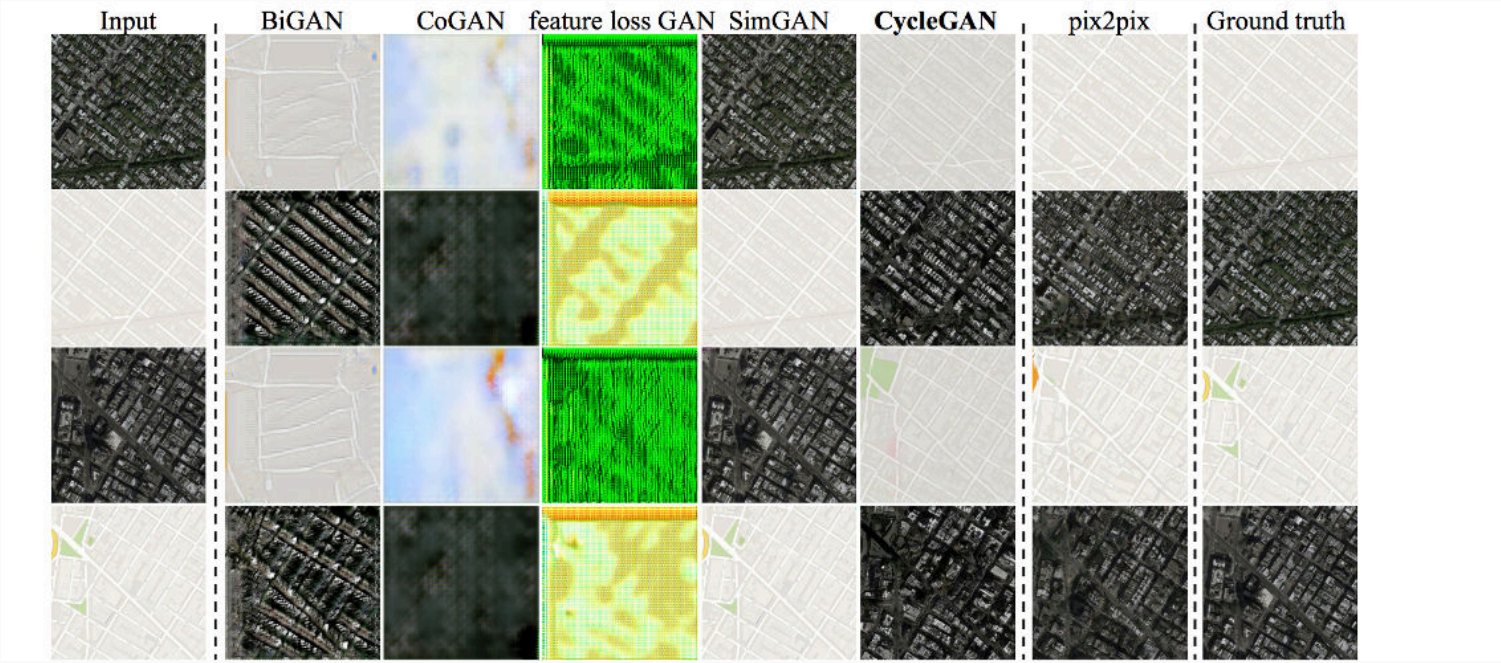
- CycleGAN (Zhu et al., 2017)
- Можно посмотреть на реконструкции тоже



CycleGAN

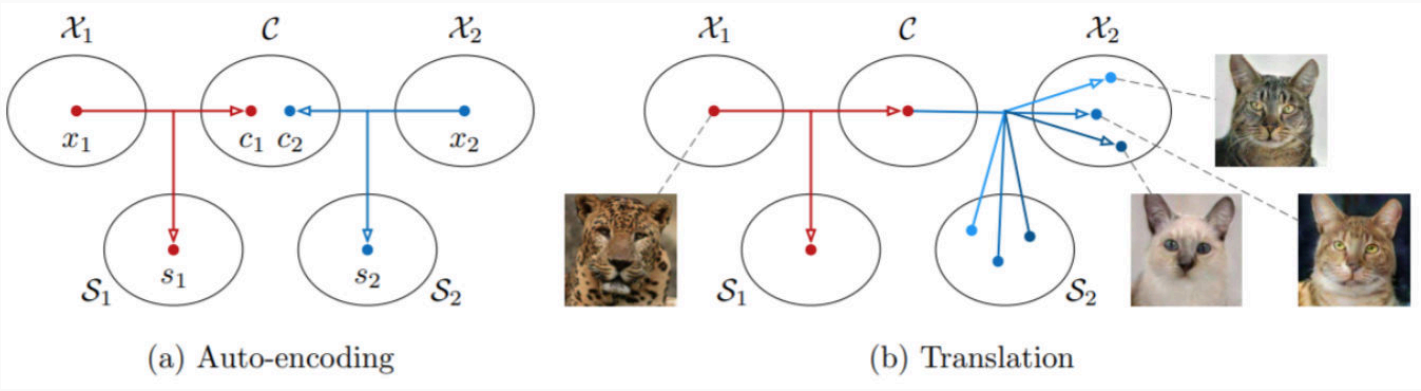
- CycleGAN (Zhu et al., 2017)
- А можно сравнить с предшественниками



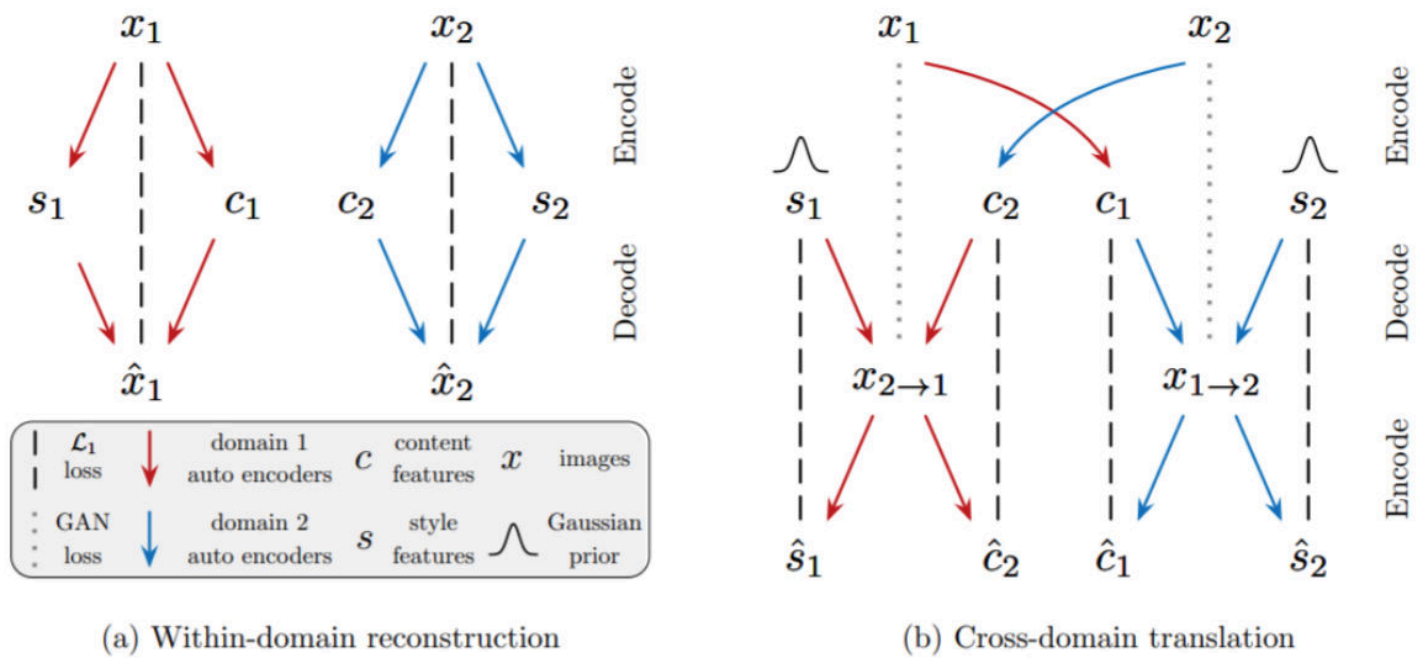


MUNIT и FUNIT

- Но всем этим моделям (кроме AdaIN) нужны большие датасеты в каждом стиле
- Следующий шаг – MUNIT (Huang et al., 2018)



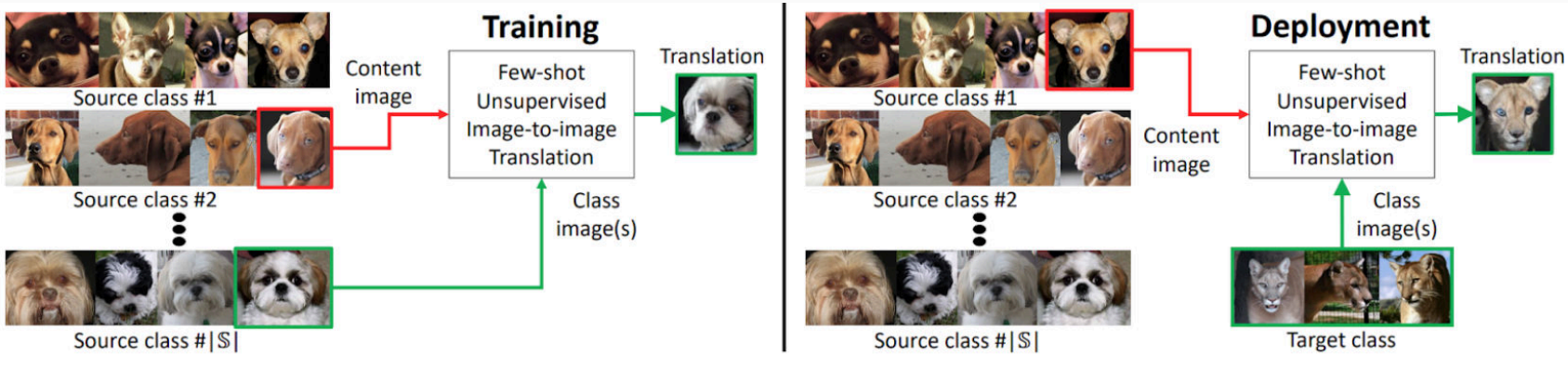
(a) Auto-encoding (b) Translation

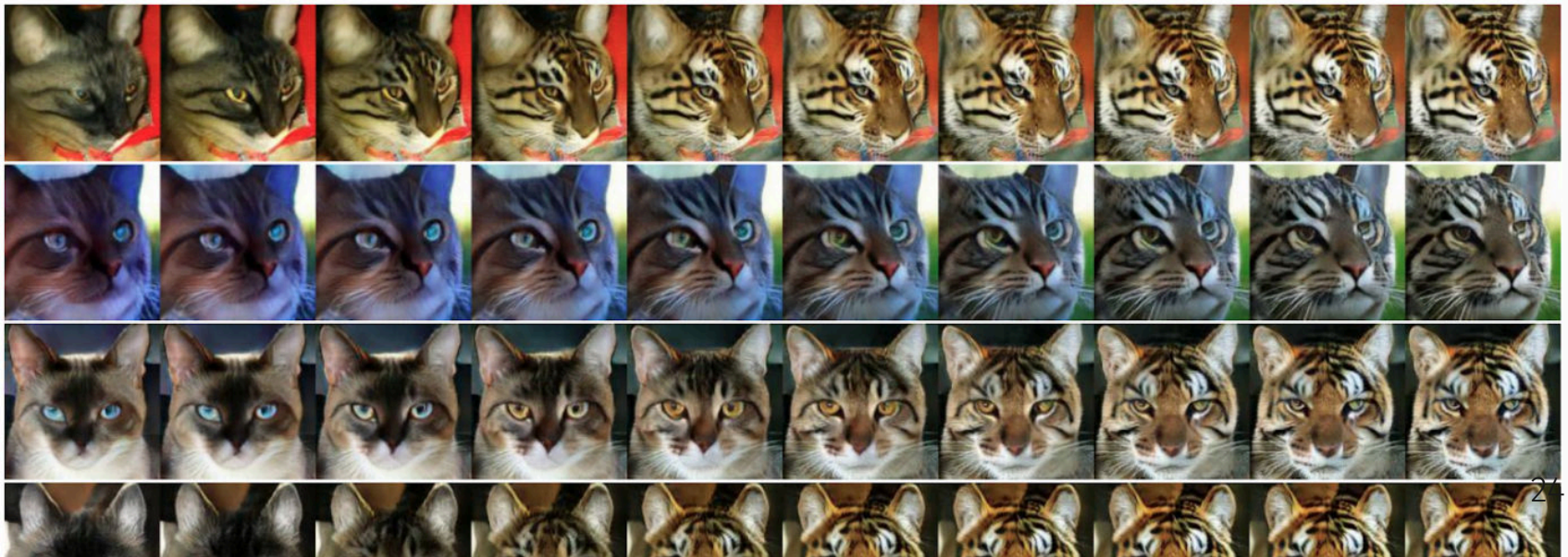


(a) Within-domain reconstruction (b) Cross-domain translation

MUNIT и FUNIT

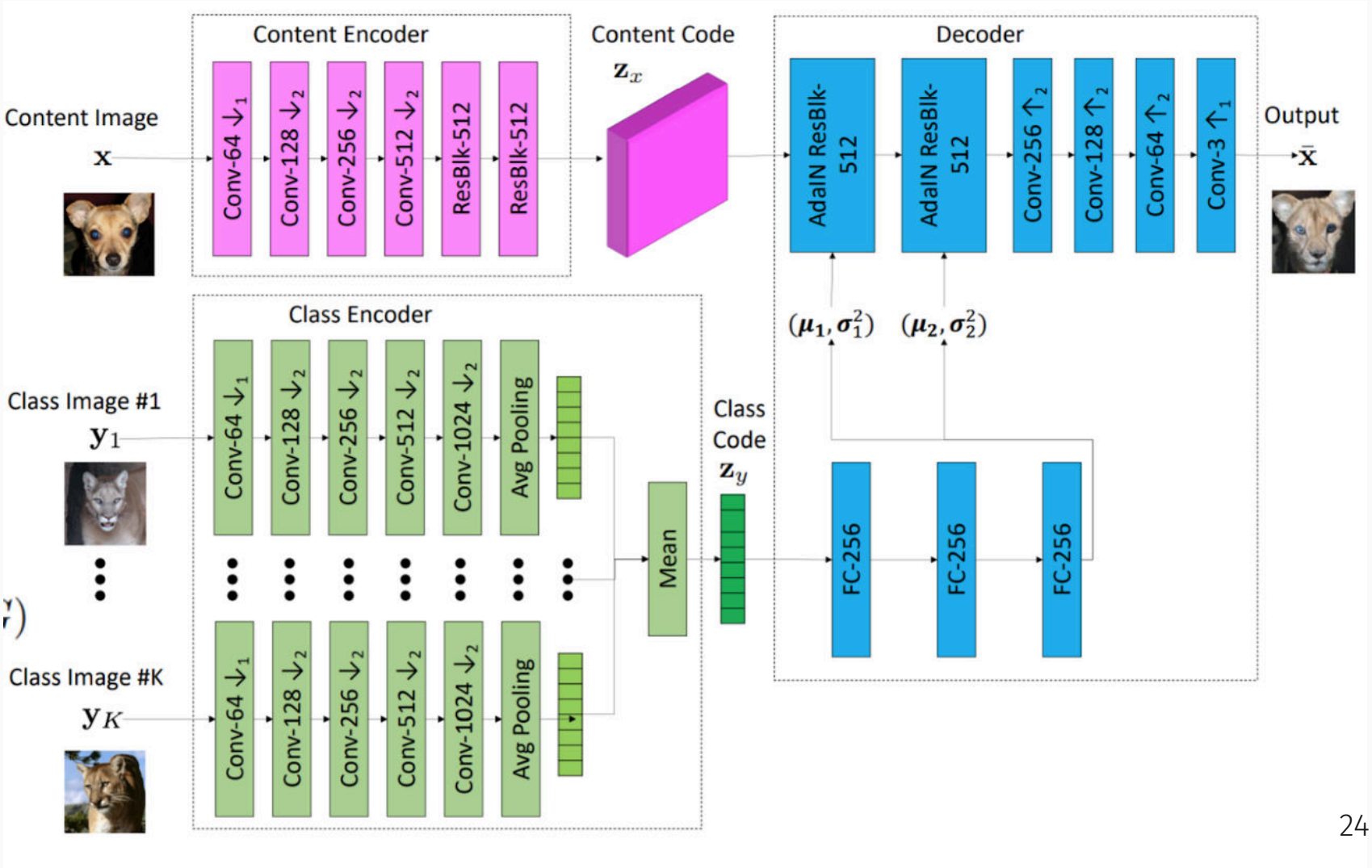
- FUNIT (Liu et al., 2019) может делать трансфер между многими стилями, определяемыми несколькими картинками





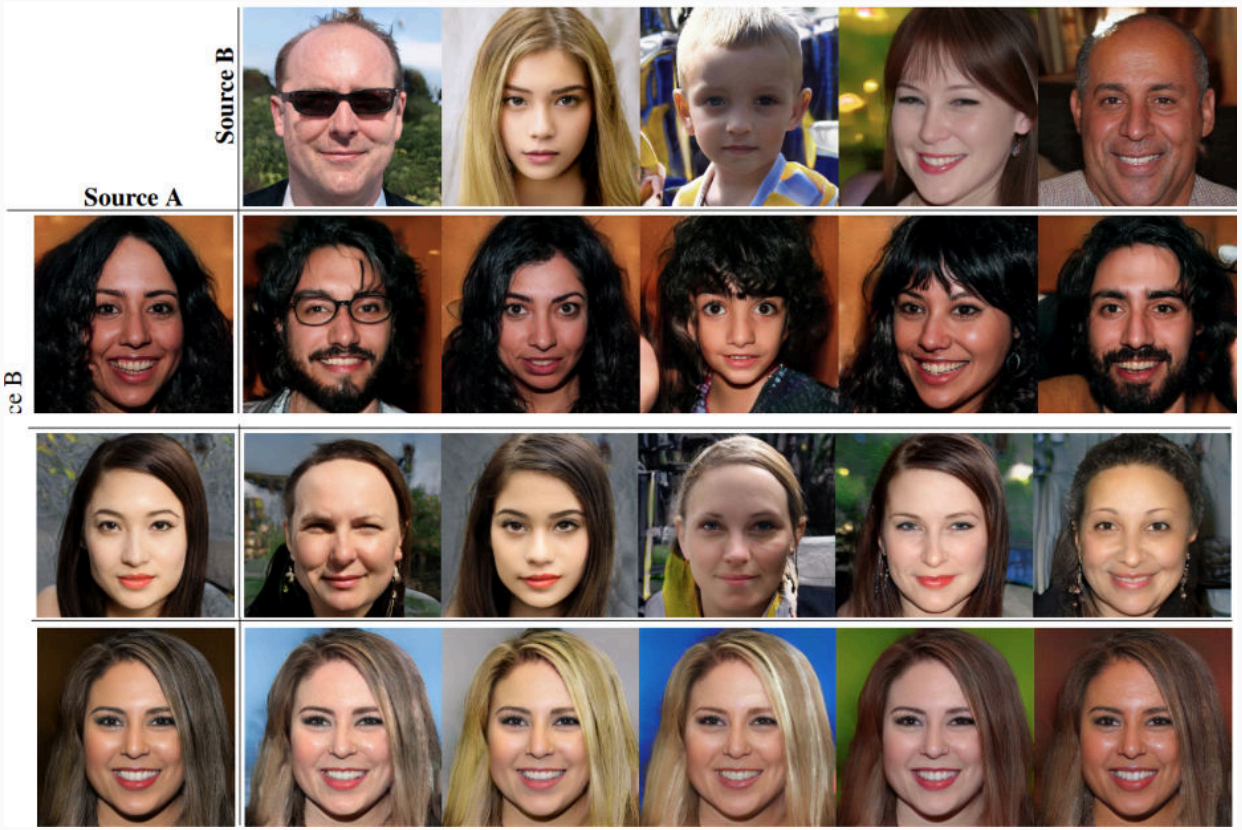
MUNIT и FUNIT

- Идея: условный генератор и multitask adversarial дискриминатор



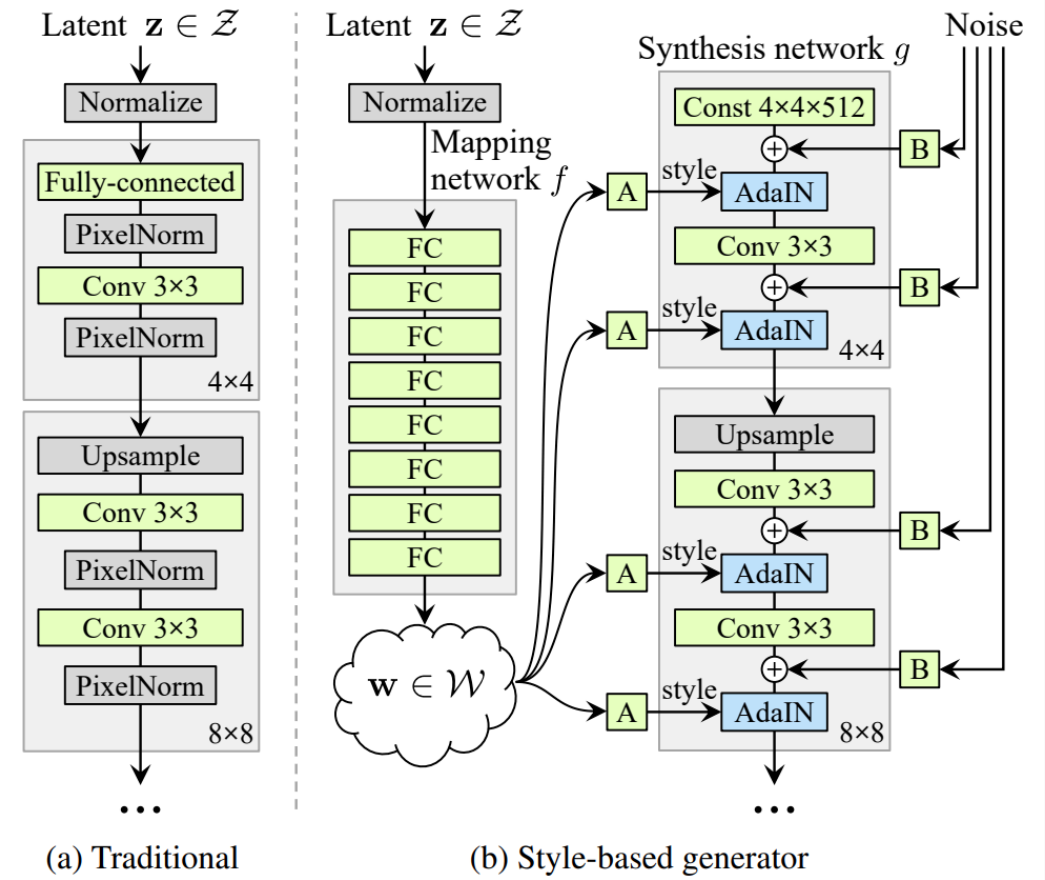
StyleGAN

- StyleGAN от NVIDIA (Karras et al., 2019) порождает лица с условиями, взятыми из других лиц:
  `https://www.youtube.com/watch?v=kSLJriaOumA`



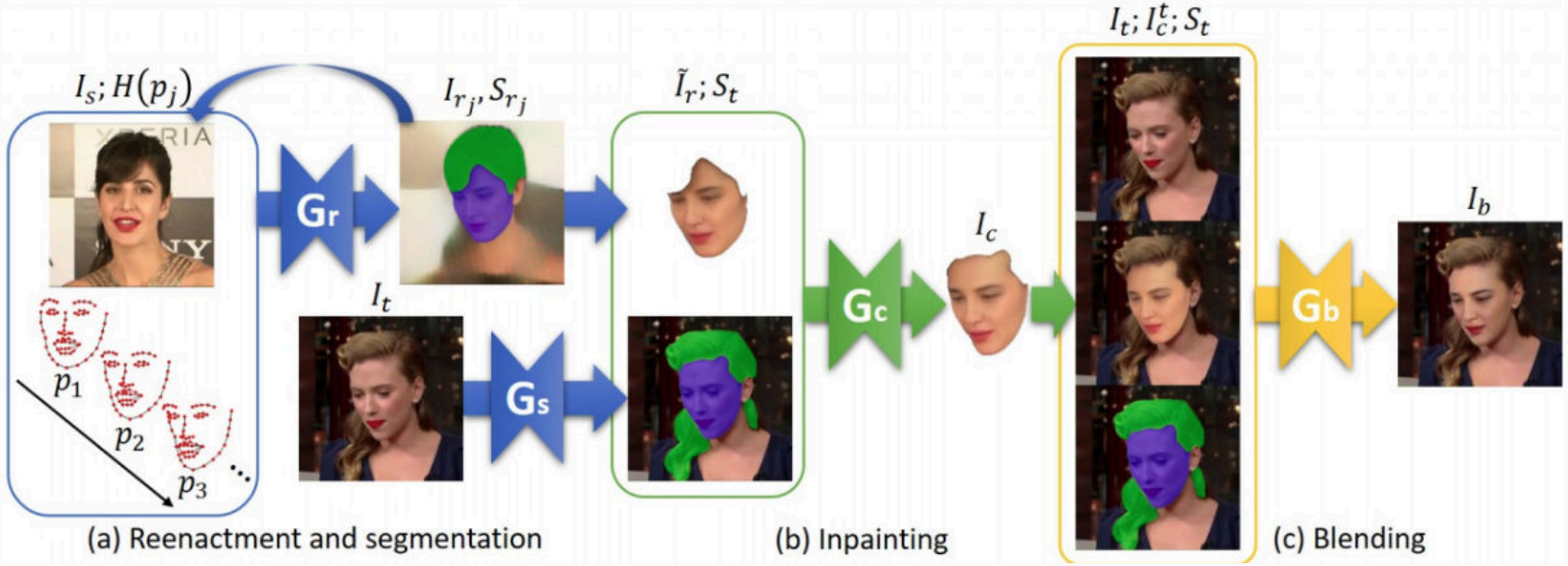
StyleGAN

- Смысл в том, чтобы подавать стиль на вход на разных уровнях генератора, и использовать его в AdaIN:



StyleGAN

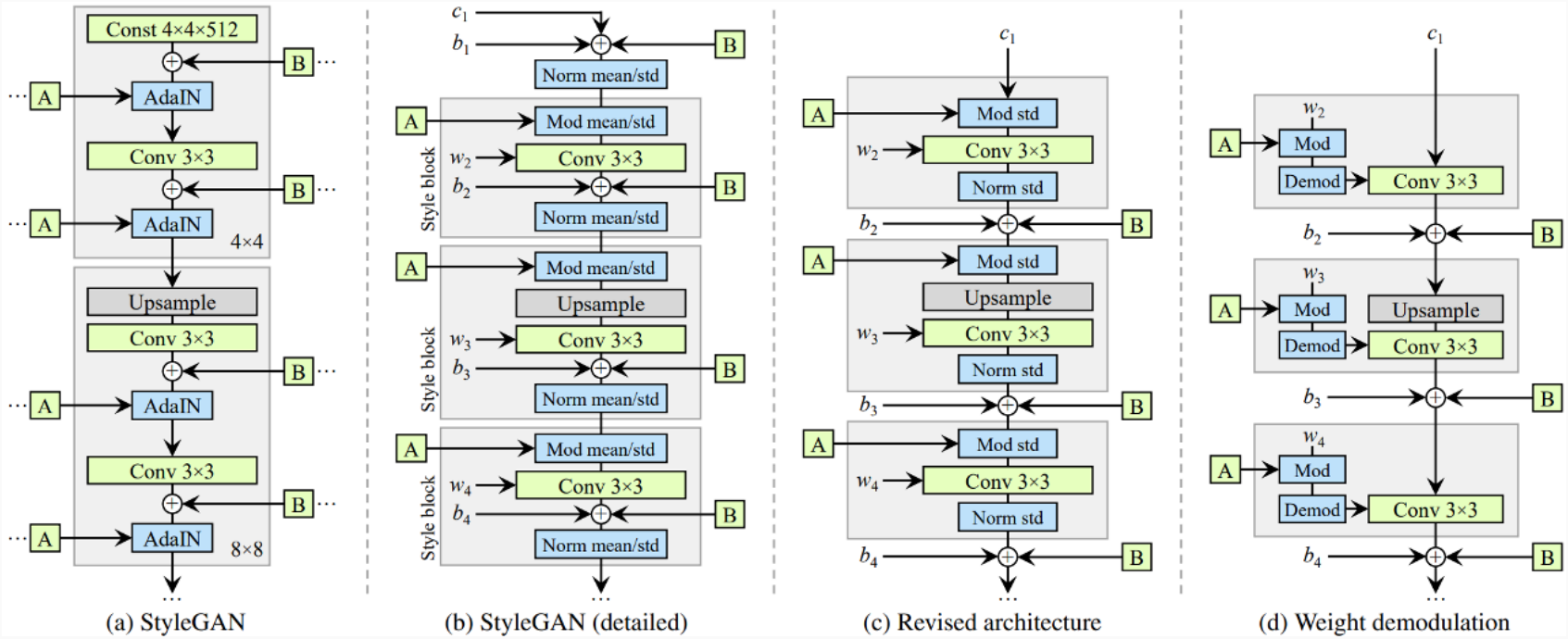
- Deepfakes тоже так работают: FSGAN (Nirkin et al., 2019)



StyleGAN

- StyleGAN2 (Karras et al., 2019) – NVIDIA смотрела на то, как дальне улучшать качество картинок, получающихся из StyleGAN
- Неочевидные проблемы – например, вот этот артефакт происходит от instance normalization:

- Поэтому они переделали архитектуры сетей в StyleGAN2:



StyleGAN

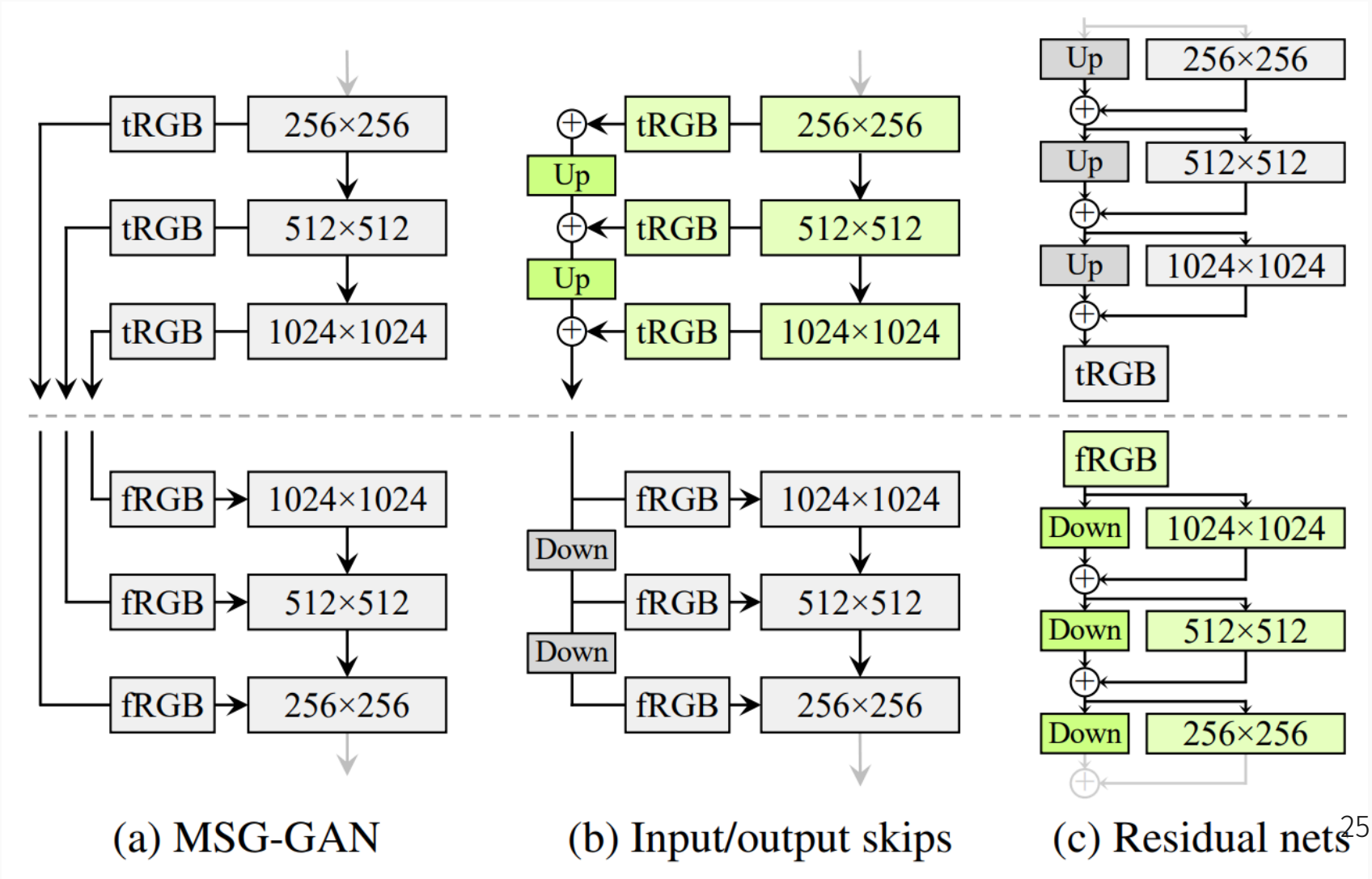
- Заменяют нормализацию на демодуляцию, т.е. чуть другое преобразование:

StyleGAN

- Есть артефакты и от progressive growing:

StyleGAN

- Поэтому вместо этого сравнивают архитектуры, которые добиваются того же эффекта внутри сети:

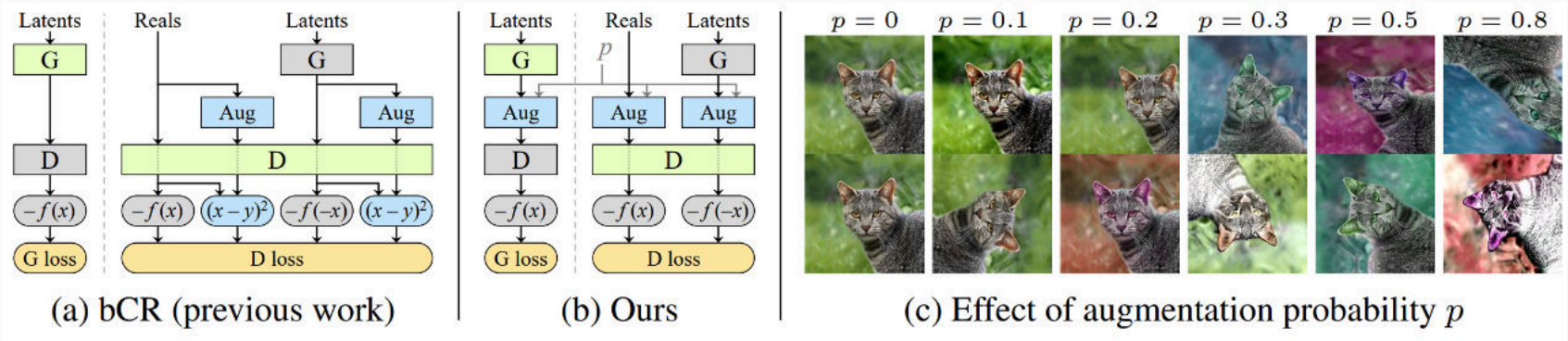


StyleGAN

- Ну и, конечно, ещё лучше получается:

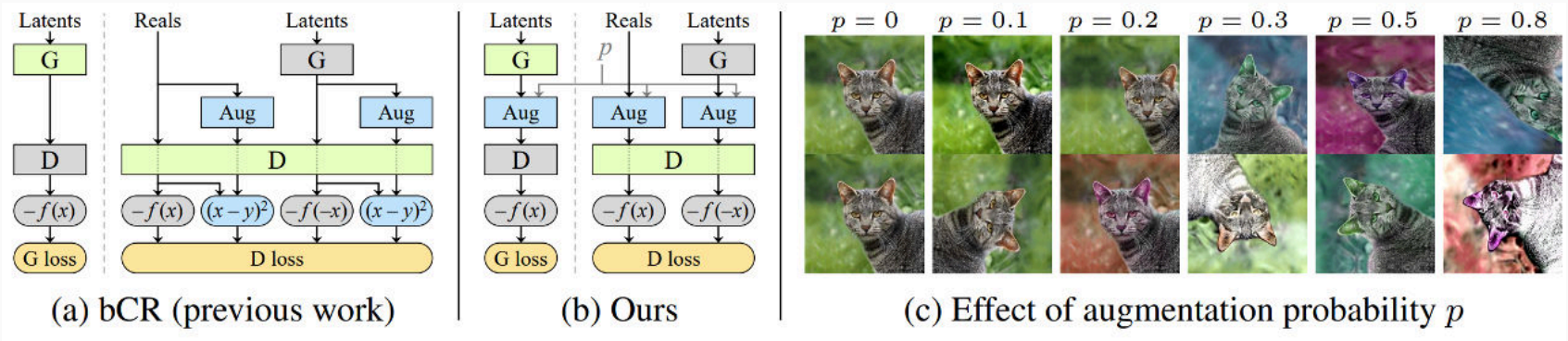
StyleGAN

- Следующая работа, (Karras et al., Oct. 2020), обращается к тому, что делать, когда данных мало
- Тут очень важны аугментации
- Раньше аугментации применялись к тому, что видит D, плюс регуляризатор на то, чтобы они не «протекали», т.е. чтобы дискриминатор не обращал внимания на аугментации



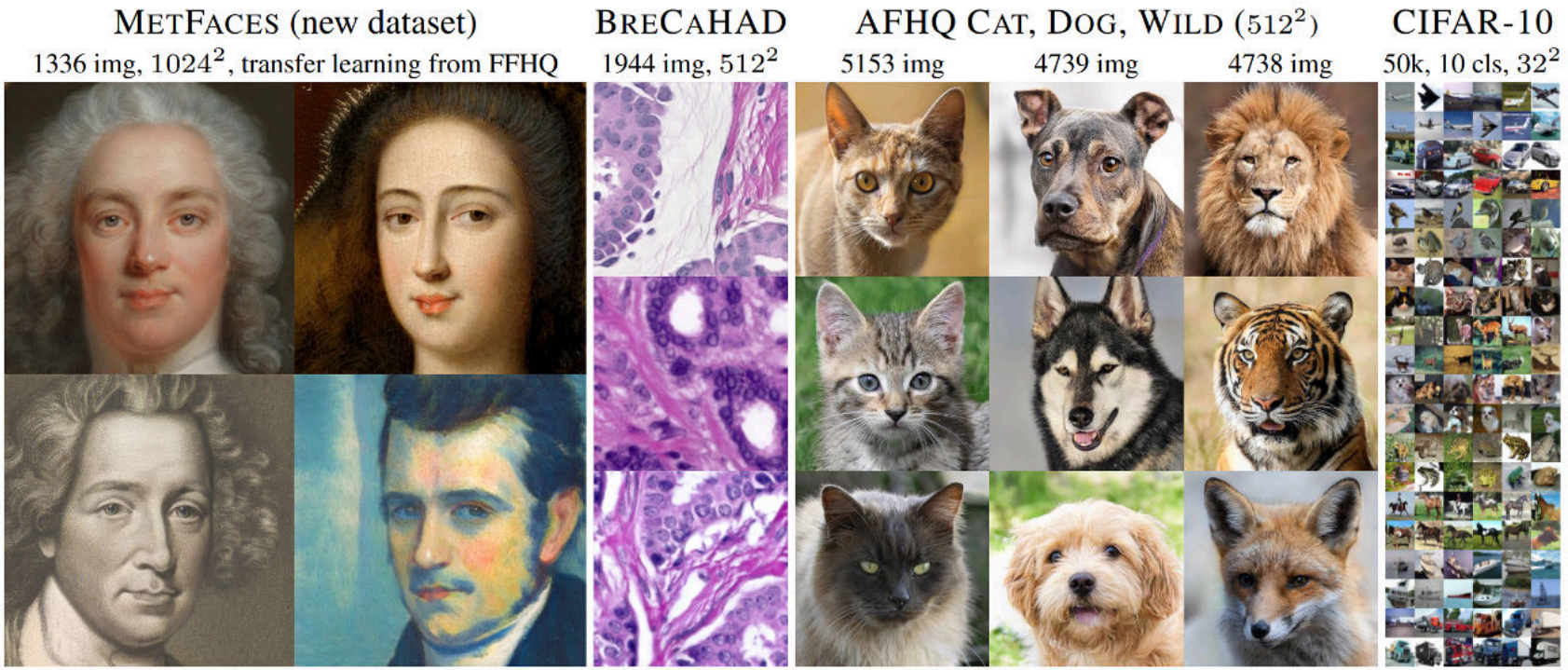
StyleGAN

- Но тогда аугментации могут «протекать» всё равно, ведь теперь для D не важно, была аугментация или нет
- Karras et al. применяют аугментации просто стохастически
- Вторая новизна – адаптивная аугментация, т.е. p настраивается динамически в зависимости от того, насколько сильный оверфиттинг мы видим



StyleGAN

- Получается отлично, особенно при transfer learning, т.е. если начать с конфигурации, обученной на другом датасете:



StyleGAN

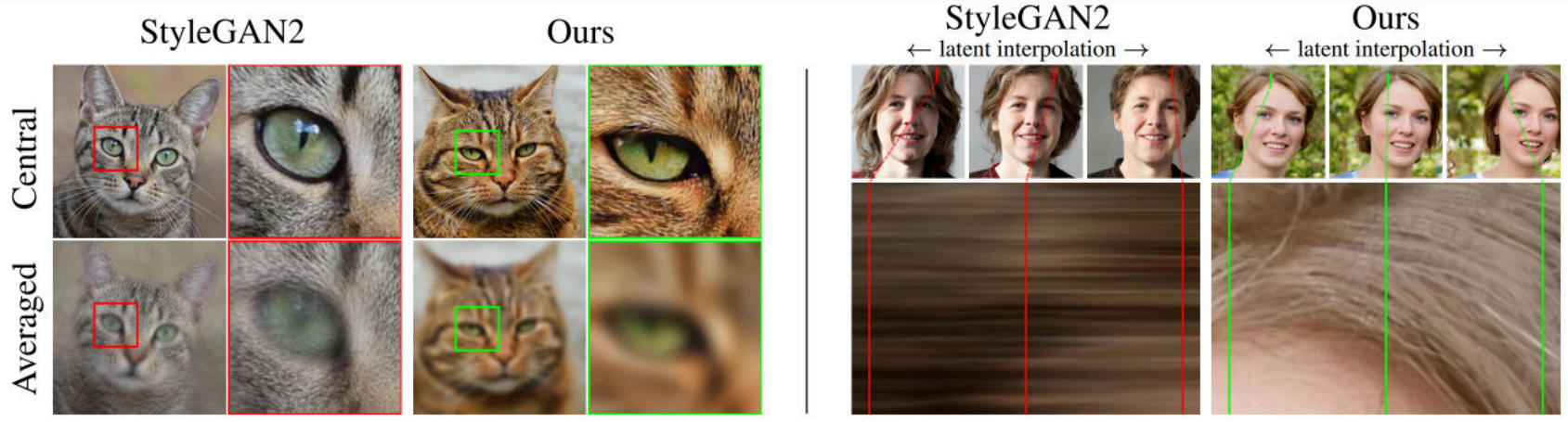
- Вот так делал StyleGAN2:

StyleGAN

- А вот так новый StyleGAN Ada:

StyleGAN

- (Karras et al., 2021): StyleGAN 3
- Улучшают latent interpolation (прогулку по пространству признаков) и борются с texture sticking (чтобы изменения в пространстве признаков были более непрерывными)



- Для этого переходят к непрерывным картам признаков! Не будем подробно об этом, но начинается DSP: нужны свёртки с дельта-функциями, чтобы перейти от непрерывного к дискретному, а потом низкочастотные фильтры, чтобы перейти обратно

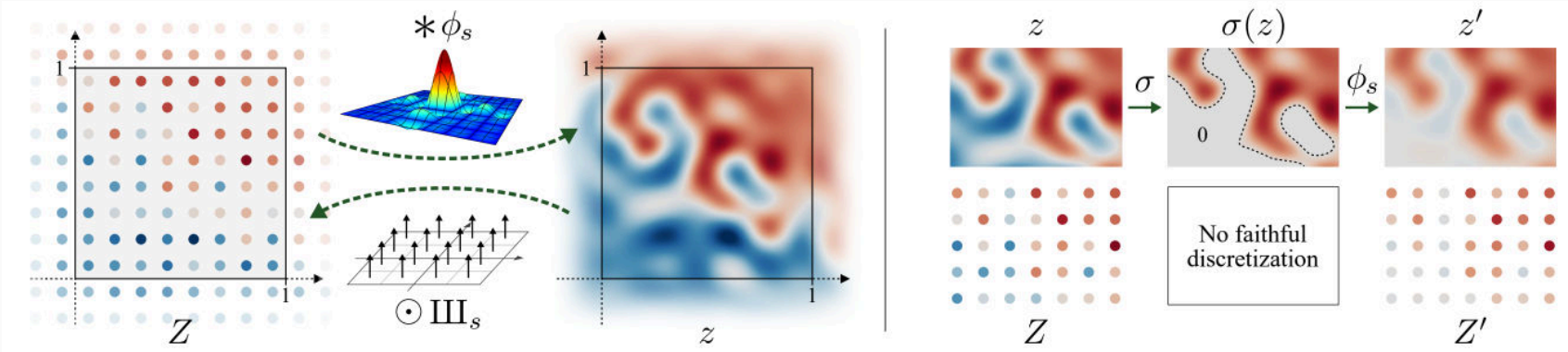
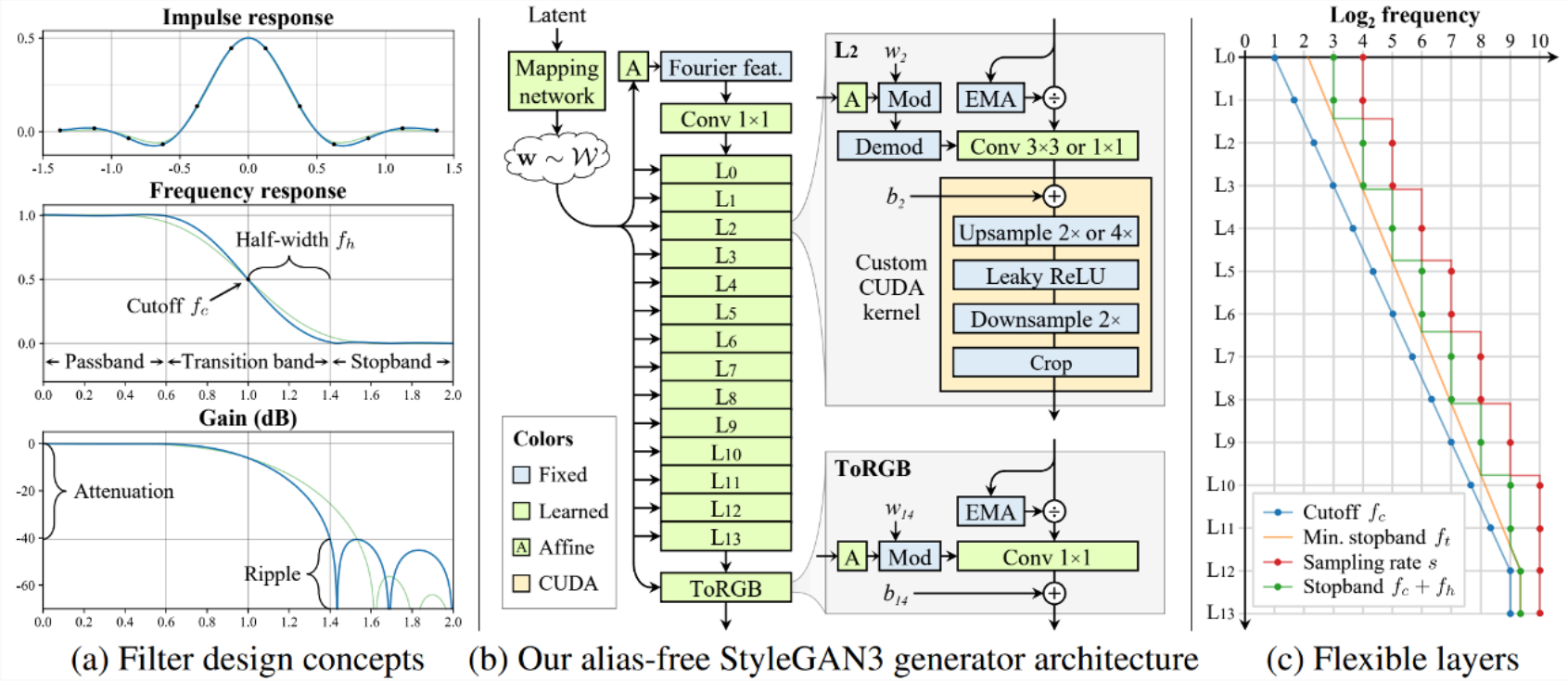


Figure 2: **Left:** Discrete representation Z and continuous representation z are related to each other via convolution with ideal interpolation filter ϕ_s and pointwise multiplication with Dirac comb Ш_s. **Right:** Nonlinearity σ, ReLU in this example, may produce arbitrarily high frequencies in the continuous-domain $\sigma(z)$. Low-pass filtering via ϕ_s is necessary to ensure that Z' captures the result.

StyleGAN

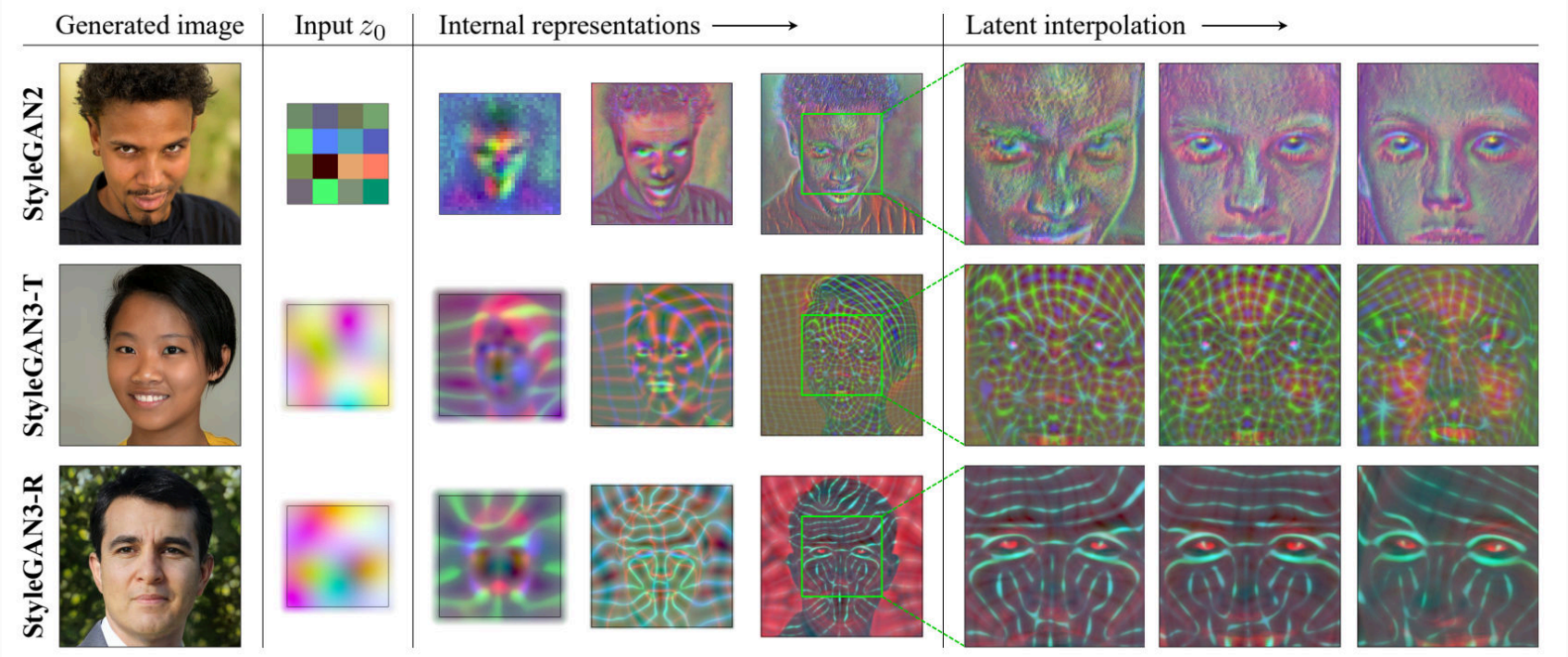
- В самой архитектуре тоже дискретные вещи заменяются на непрерывные



(a) Filter design concepts (b) Our alias-free StyleGAN3 generator architecture (c) Flexible layers

StyleGAN

- В результате получается, что некоторые признаки кодируют фазу, а не только амплитуду сигналов



Спасибо за внимание!