курс «Глубокое обучение»

Генеративные модели: итоги

Александр Дьяконов

16 декабря 2021 года

План

Conditional GANs: Text-to-Image Synthesis

StackGAN: описание $\rightarrow$ изображение

Задача трансляции изображений

MUNIT: идея пространств контента и стиля

Instance Normalization

FUNIT (NVIDIA, 2019)

Немного про перенос стиля: AdaIN

StyleGAN – перенос стиля

StyleGAN2: избавление от артефактов

Conditional GANs: Text-to-Image Synthesis

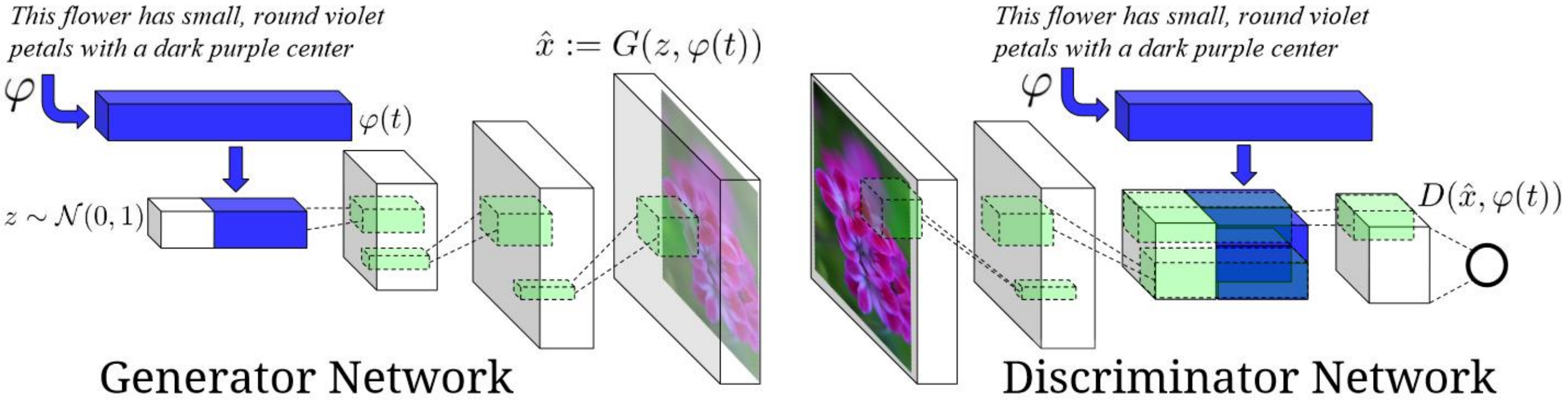


Figure 2. Our text-conditional convolutional GAN architecture. Text encoding $\varphi(t)$ is used by both generator and discriminator. It is projected to a lower-dimensions and depth concatenated with image feature maps for further stages of convolutional processing.

отдельная задача – классификатор соответствия (pix, text) для обучения представлений
негативный пример: любая несоответствующая пара (изображение, текст)

Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., & Lee, H. «Generative adversarial text to image synthesis» // ICML https://arxiv.org/pdf/1605.05396.pdf

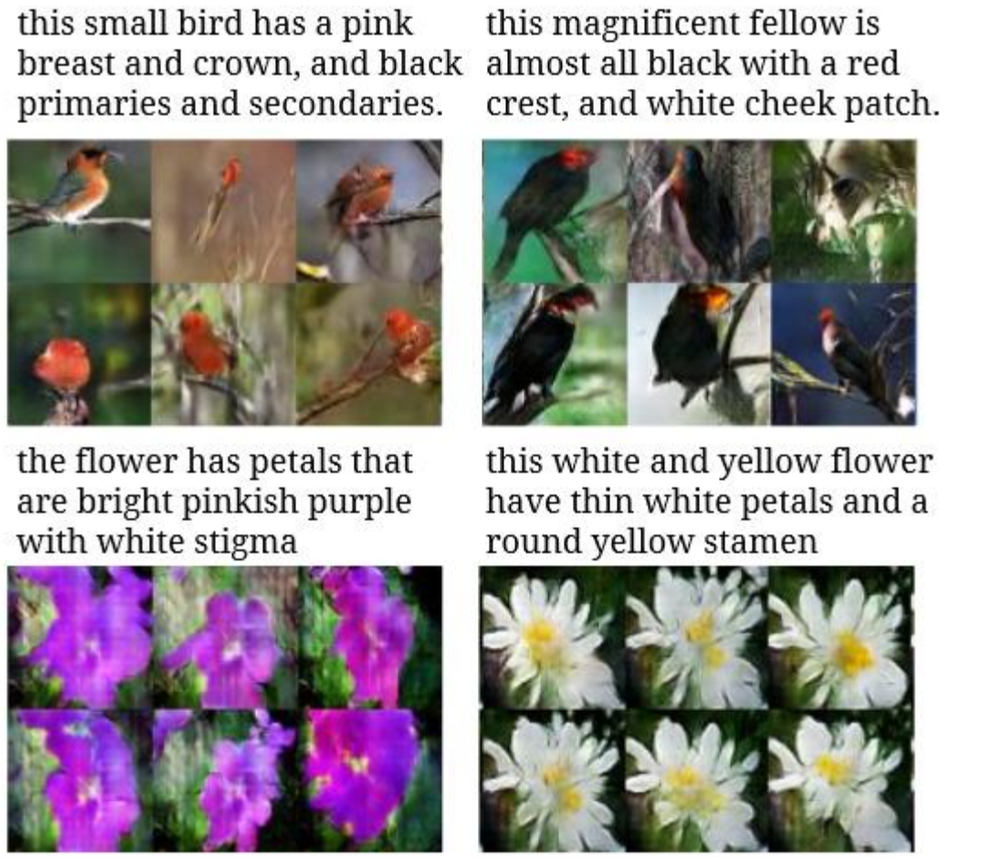


Figure 1. Examples of generated images from text descriptions. Left: captions are from zero-shot (held out) categories, unseen text. Right: captions are from the training set.

Отдельно выучивается style encoder network, которая по изображению восстанавливает z

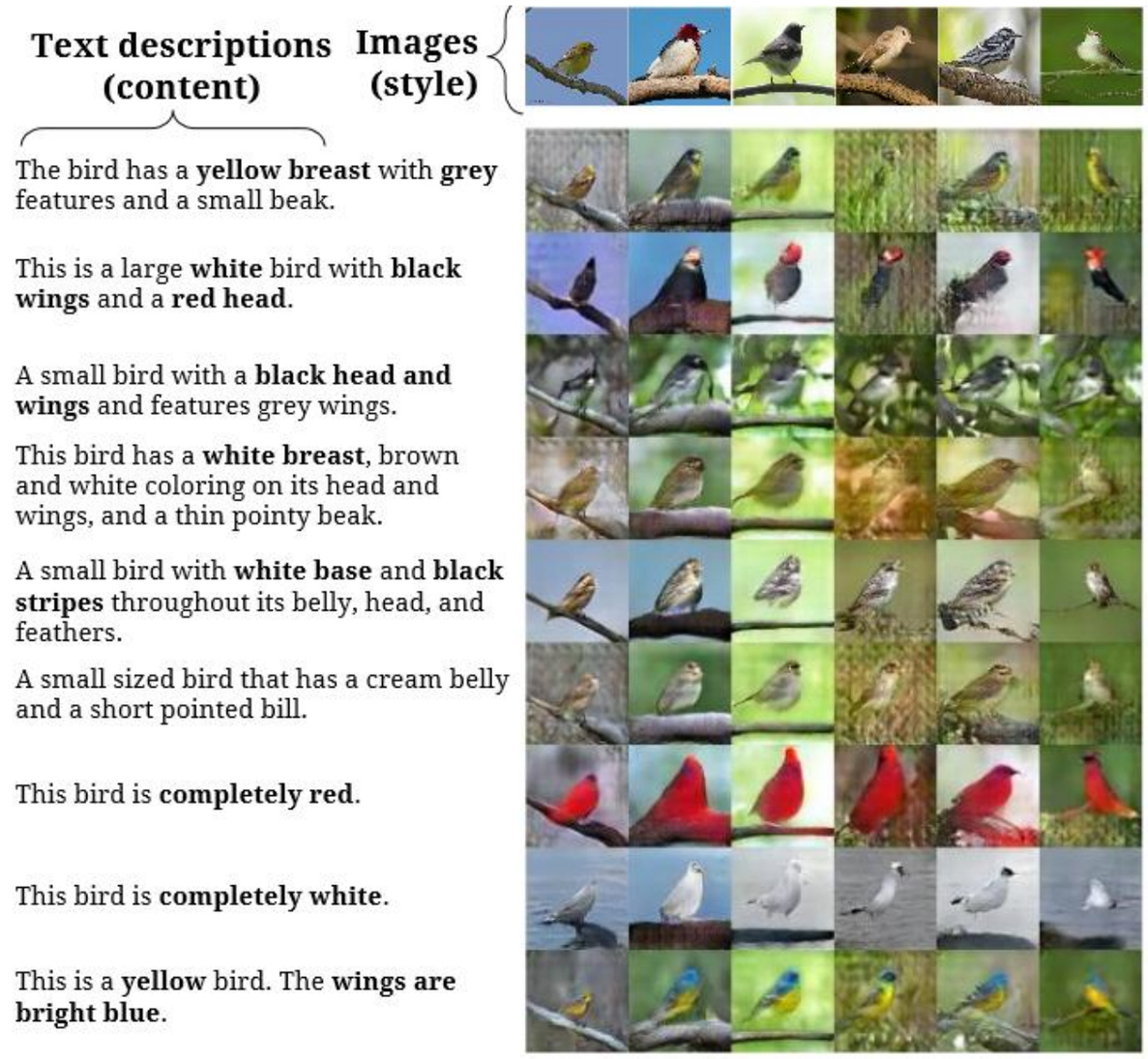


Figure 6. Transfering style from the top row (real) images to the content from the query text, with G acting as a deterministic decoder. The bottom three rows are captions made up by us.

Conditional GANs: Text-to-Image Synthesis

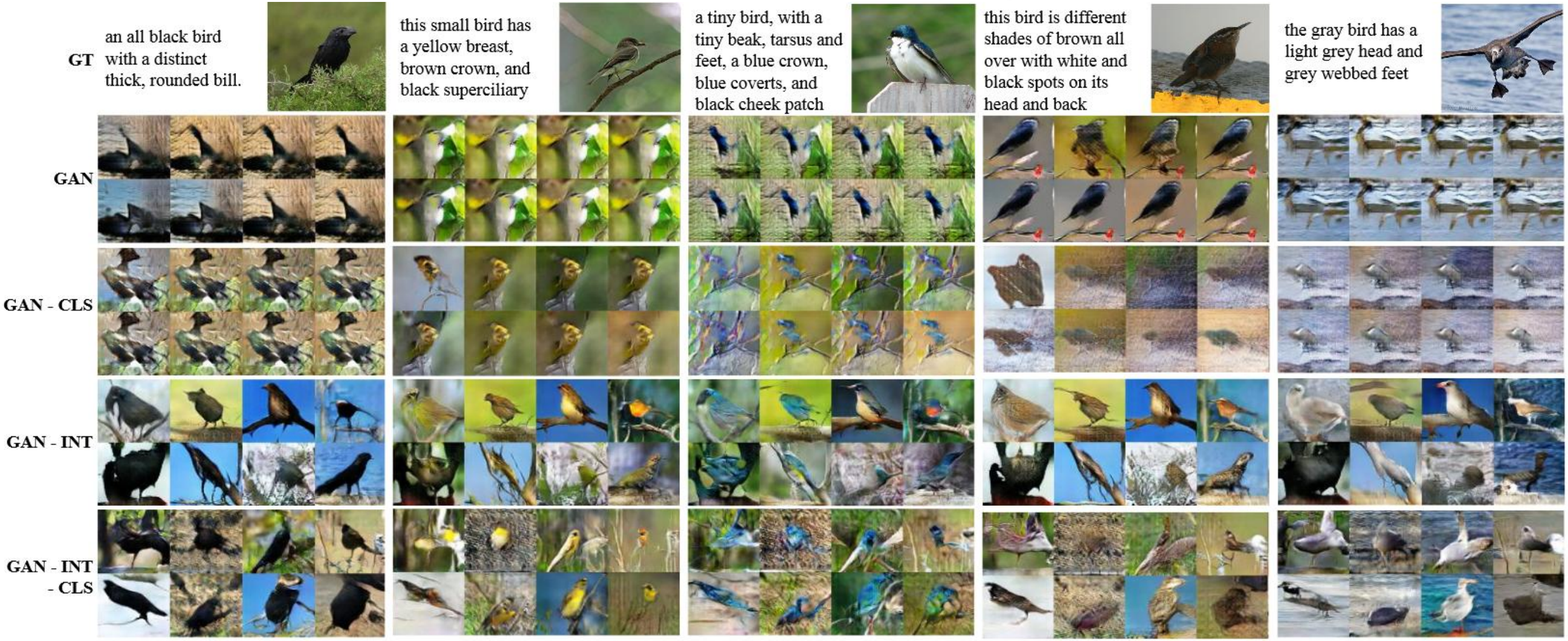


Figure 3. Zero-shot (i.e. conditioned on text from unseen test set categories) generated bird images using GAN, GAN-CLS, GAN-INT and GAN-INT-CLS. We found that interpolation regularizer was needed to reliably achieve visually-plausible results.

тут разные версии GANов из статьи

Conditional GANs: Text-to-Image Synthesis

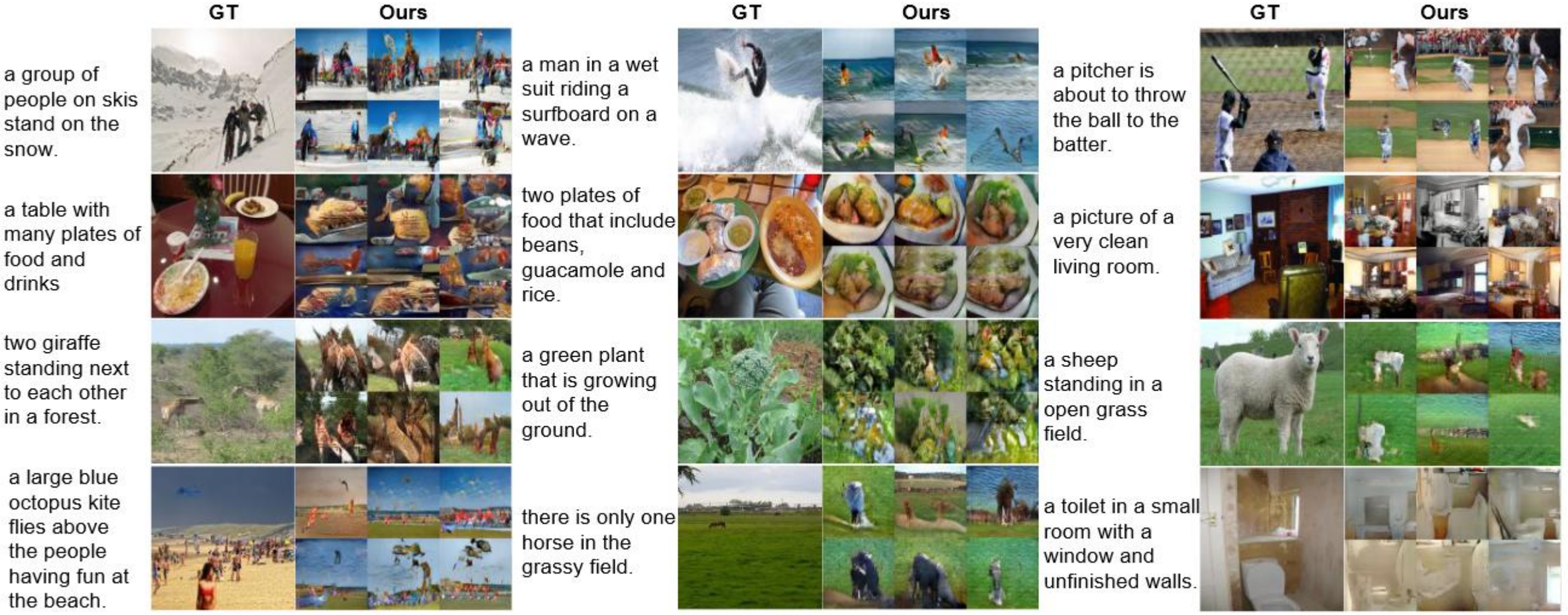


Figure 7. Generating images of general concepts using our GAN-CLS on the MS-COCO validation set. Unlike the case of CUB and Oxford-102, the network must (try to) handle multiple objects and diverse backgrounds.

StackGAN: описание → изображение

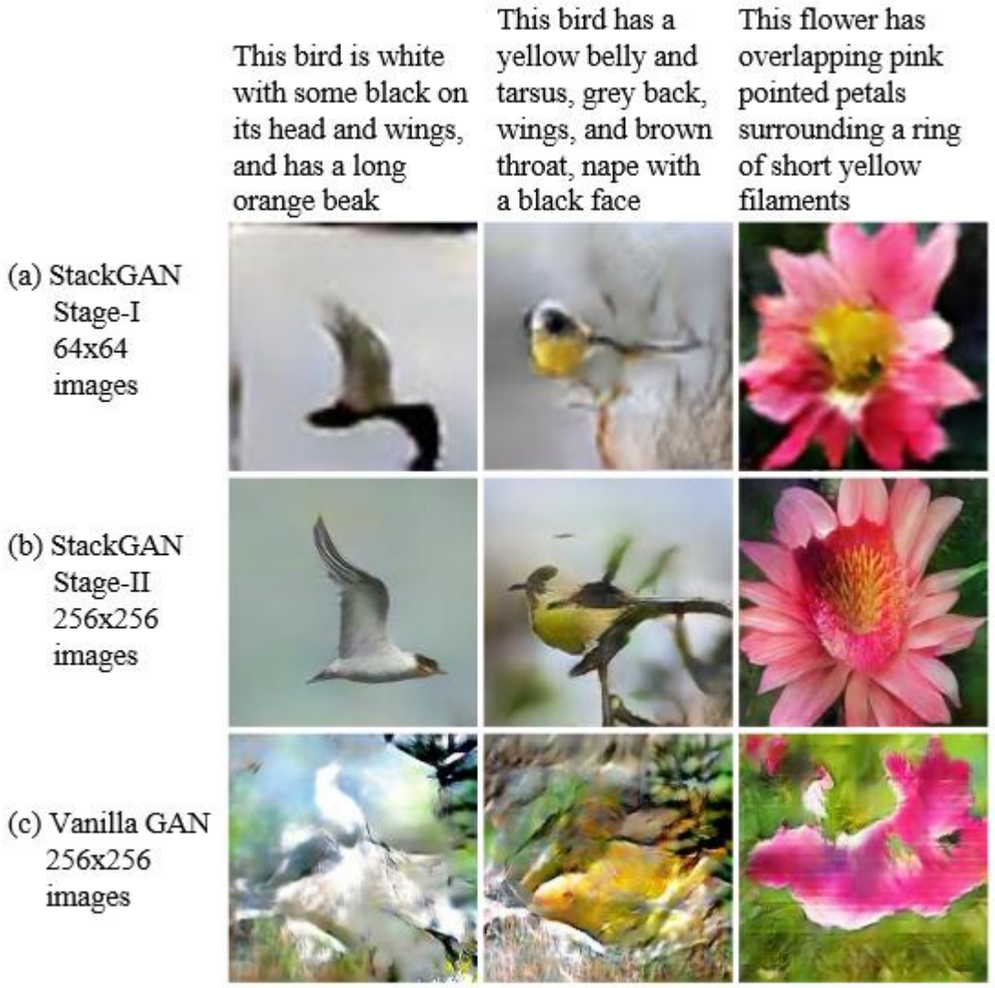


Figure 1. Comparison of the proposed StackGAN and a vanilla one-stage GAN for generating 256×256 images. (a) Given text descriptions, Stage-I of StackGAN sketches rough shapes and basic colors of objects, yielding low-resolution images. (b) Stage-II of StackGAN takes Stage-I results and text descriptions as inputs, and generates high-resolution images with photo-realistic details. (c) Results by a vanilla 256×256 GAN which simply adds more upsampling layers to state-of-the-art GAN-INT-CLS [26]. It is unable to generate any plausible images of 256×256 resolution.

Han Zhang «StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks» // https://arxiv.org/abs/1612.03242

StackGAN: описание → изображение

Два этапа:

Stage-I GAN: «сырое изображение»

низкое разрешение, примитивная форма, цвета

Stage-II GAN: окончательное

Conditioning Augmentation

идея: используем не прямо представление текста,

а немного зашумляем

Figure 7. Conditioning Augmentation (CA) helps stabilize the training of conditional GAN and improves the diversity of the generated samples. (Row 1) without CA, Stage-I GAN fails to generate plausible 256×256 samples. Although different noise vector z is used for each column, the generated samples collapse to be the same for each input text description. (Row 2-3) with CA but fixing the noise vectors z, methods are still able to generate birds with different poses and viewpoints.

StackGAN: описание → изображение

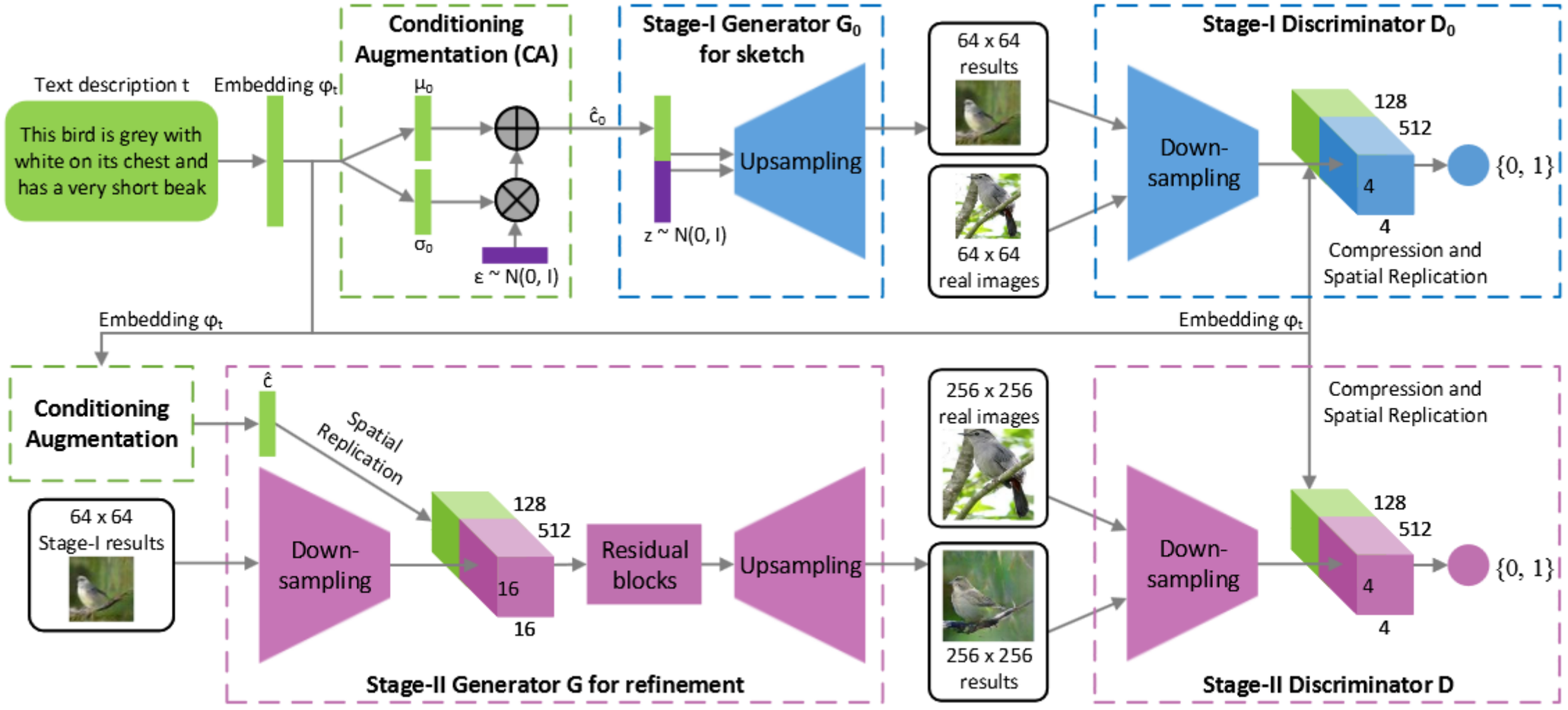


Figure 2. The architecture of the proposed StackGAN. The Stage-I generator draws a low-resolution image by sketching rough shape and basic colors of the object from the given text and painting the background from a random noise vector. Conditioned on Stage-I results, the Stage-II generator corrects defects and adds compelling details into Stage-I results, yielding a more realistic high-resolution image.

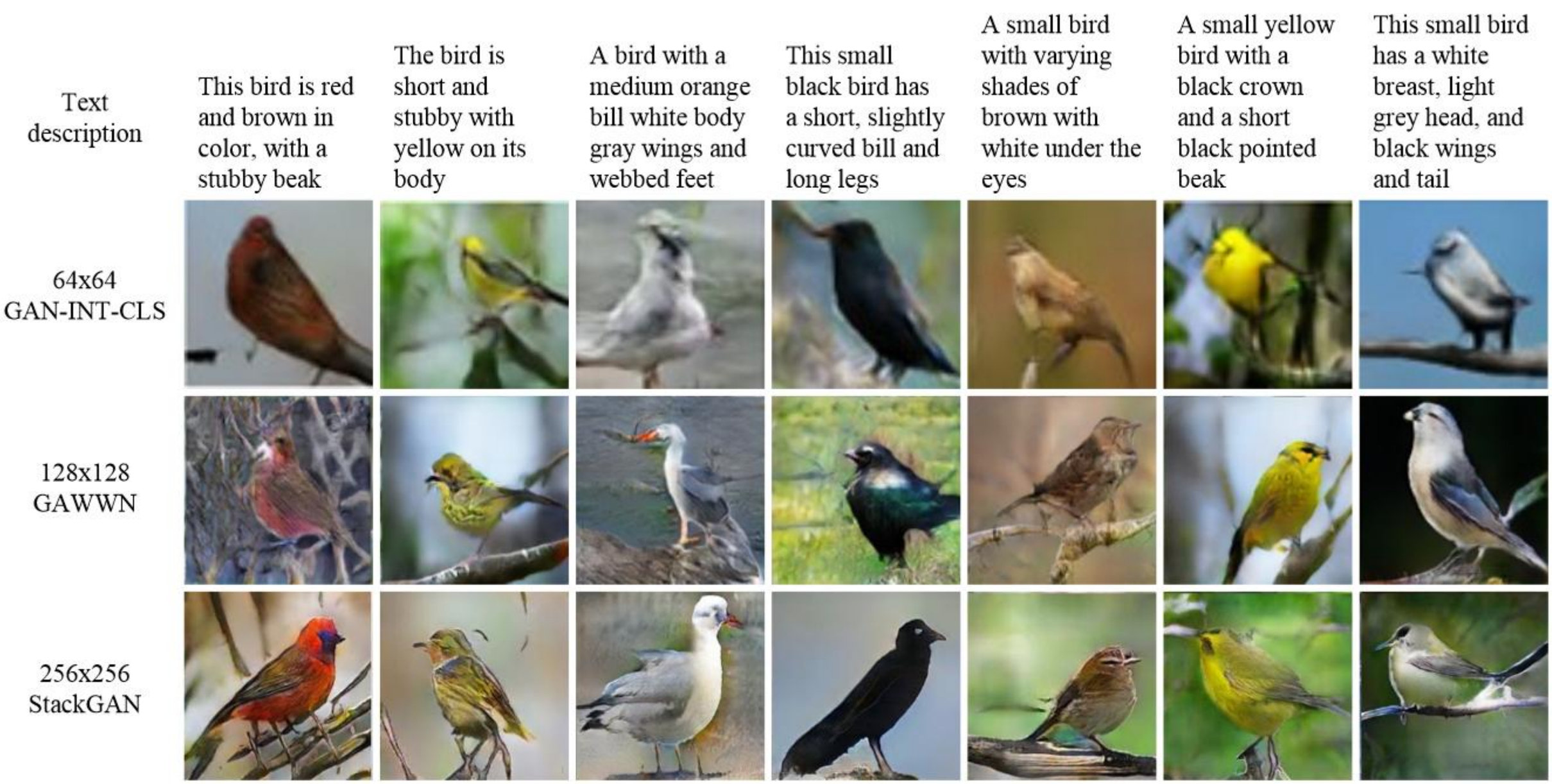
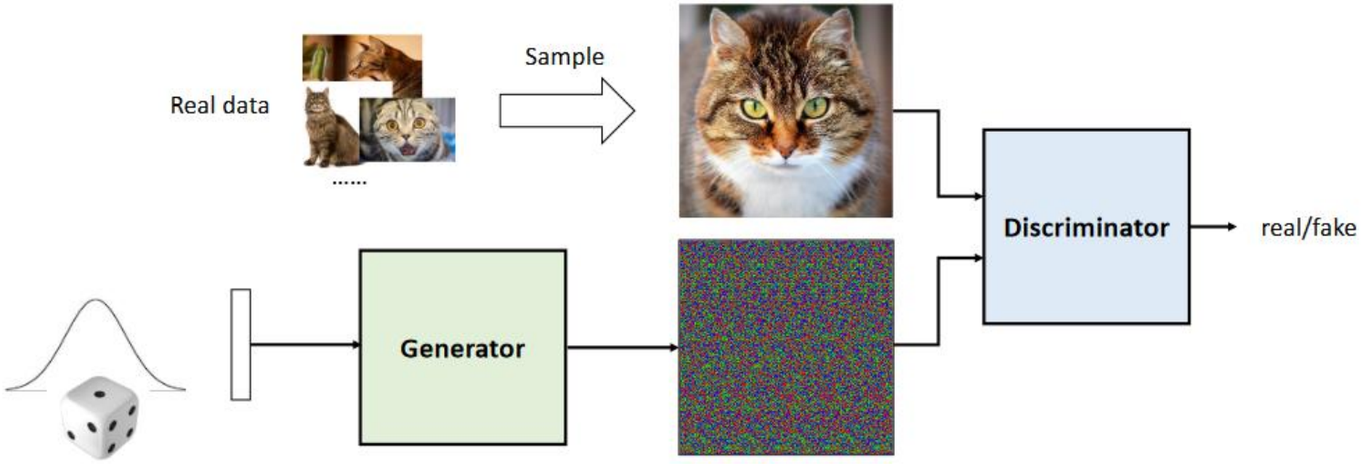


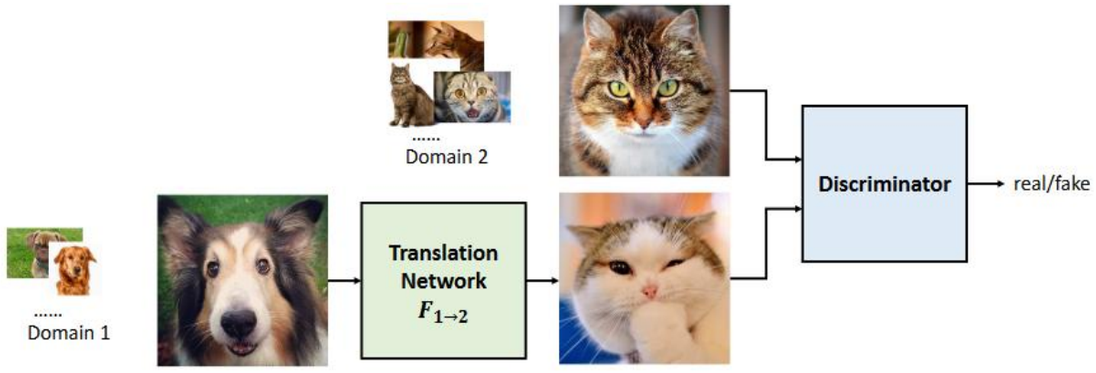
Figure 3. Example results by our StackGAN, GAWWN [24], and GAN-INT-CLS [26] conditioned on text descriptions from CUB test set.

Задача трансляции изображений

Обычный GAN

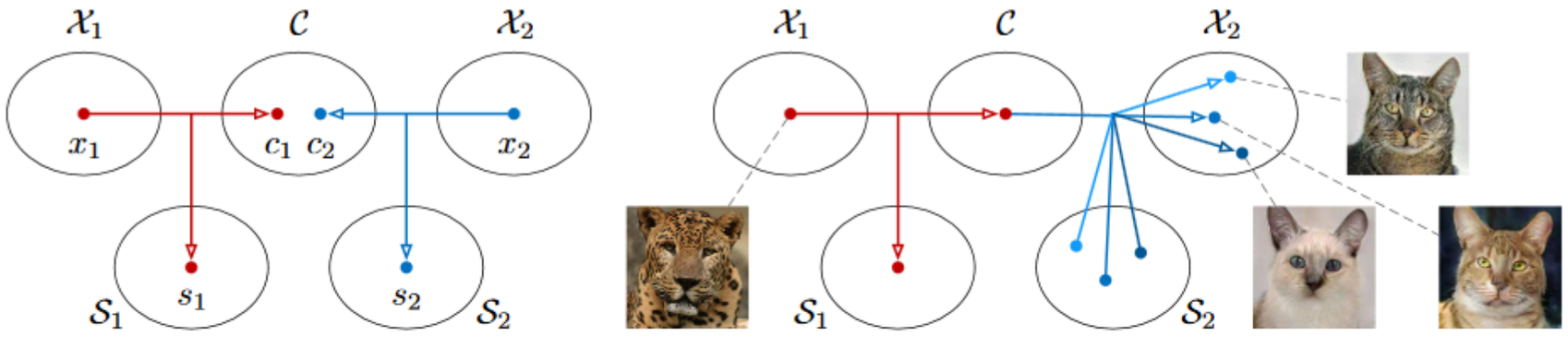


GAN для Unsupervised Image-to-Image Translation



https://mingyuliutw.github.io/icip2019-image-translation-tutorial/pdfs/part4_munit.pdf

MUNIT: идея пространств контента и стиля



(a) Auto-encoding (b) Translation

Fig. 1. An illustration of our method. (a) Images in each domain $\mathcal{X}_i$ are encoded to a shared content space $\mathcal{C}$ and a domain-specific style space $\mathcal{S}_i$. Each encoder has an inverse decoder omitted from this figure. (b) To translate an image in $\mathcal{X}_1$ (*e.g.*, a leopard) to $\mathcal{X}_2$ (*e.g.*, domestic cats), we recombine the content code of the input with a random style code in the target style space. Different style codes lead to different outputs.

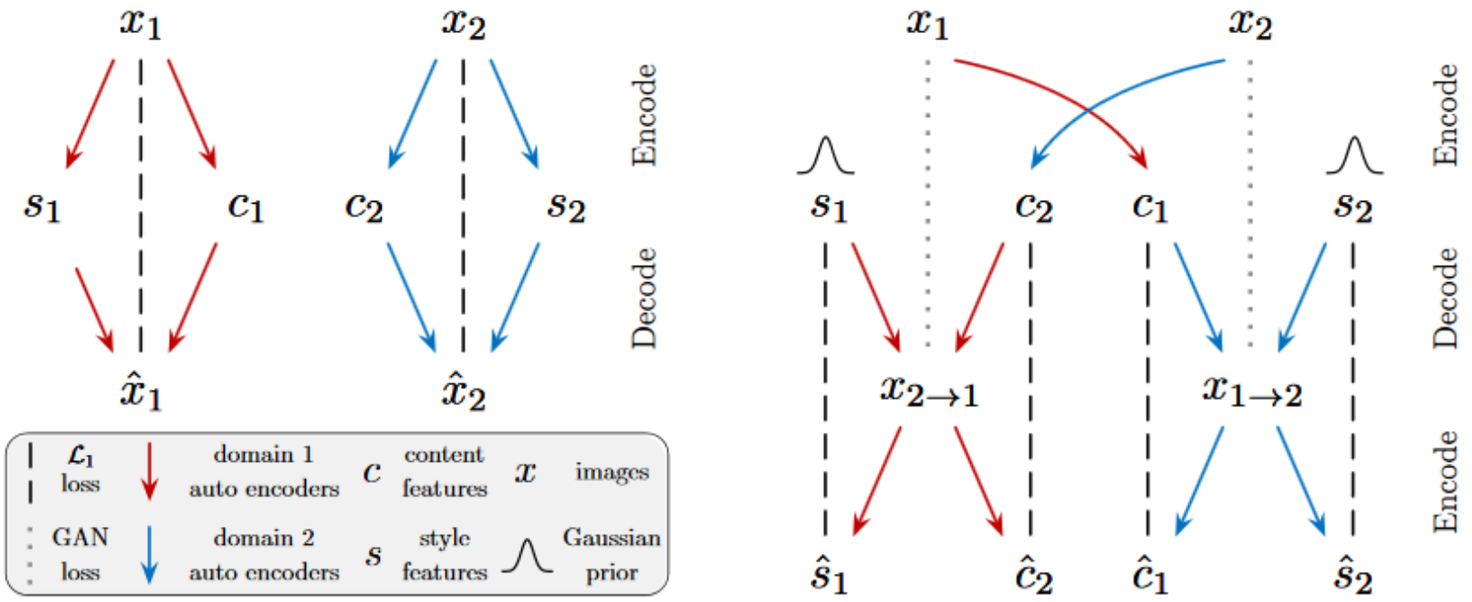
трансляция изображения в целевое распределение

нет попарной выборки

Xun Huang «Multimodal Unsupervised Image-to-Image Translation»

https://arxiv.org/pdf/1804.04732.pdf

MUNIT: какие здесь loss-функции



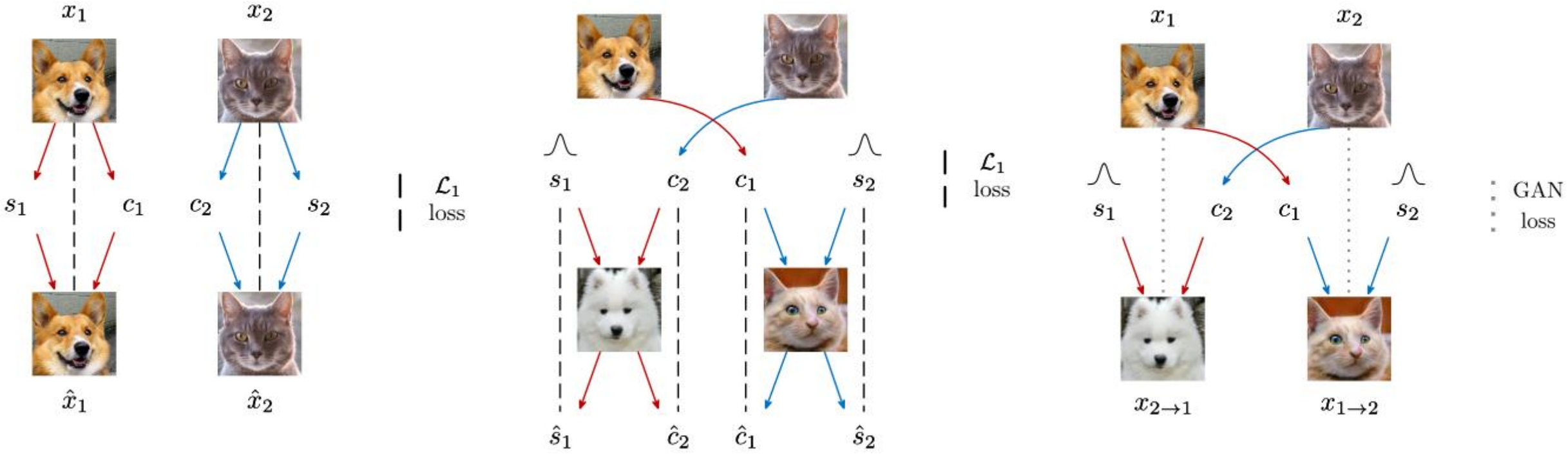
(a) Within-domain reconstruction (b) Cross-domain translation

Fig. 2. Model overview. Our image-to-image translation model consists of two auto-encoders (denoted by red and blue arrows respectively), one for each domain. The latent code of each auto-encoder is composed of a content code c and a style code s. We train the model with adversarial objectives (dotted lines) that ensure the translated images to be indistinguishable from real images in the target domain, as well as bidirectional reconstruction objectives (dashed lines) that reconstruct both images and latent codes.

latent space = content space + style space

content space – общее для всех изображений, style space – нет

MUNIT: какие здесь loss-функции



MUNIT

– **Image reconstruction.** Given an image sampled from the data distribution, we should be able to reconstruct it after encoding and decoding.

$$\mathcal{L}_{\text{recon}}^{x_1} = \mathbb{E}_{x_1 \sim p(x_1)}[||G_1(E_1^c(x_1), E_1^s(x_1)) - x_1||_1] \quad (1)$$

– **Latent reconstruction.** Given a latent code (style and content) sampled from the latent distribution at translation time, we should be able to reconstruct it after decoding and encoding.

$$\mathcal{L}_{\text{recon}}^{c_1} = \mathbb{E}_{c_1 \sim p(c_1), s_2 \sim q(s_2)}[||E_2^c(G_2(c_1, s_2)) - c_1||_1] \quad (2)$$

$$\mathcal{L}_{\text{recon}}^{s_2} = \mathbb{E}_{c_1 \sim p(c_1), s_2 \sim q(s_2)}[||E_2^s(G_2(c_1, s_2)) - s_2||_1] \quad (3)$$

where $q(s_2)$ is the prior $\mathcal{N}(0, \mathbf{I})$, $p(c_1)$ is given by $c_1 = E_1^c(x_1)$ and $x_1 \sim p(x_1)$.

Adversarial loss. We employ GANs to match the distribution of translated images to the target data distribution. In other words, images generated by our model should be indistinguishable from real images in the target domain.

$$\mathcal{L}_{\text{GAN}}^{x_2} = \mathbb{E}_{c_1 \sim p(c_1), s_2 \sim q(s_2)}[\log(1 - D_2(G_2(c_1, s_2)))] + \mathbb{E}_{x_2 \sim p(x_2)}[\log D_2(x_2)] \quad (4)$$

where D_2 is a discriminator that tries to distinguish between translated images and real images in $\mathcal{X}_2$. The discriminator D_1 and loss $\mathcal{L}_{\text{GAN}}^{x_1}$ are defined similarly.

MUNIT

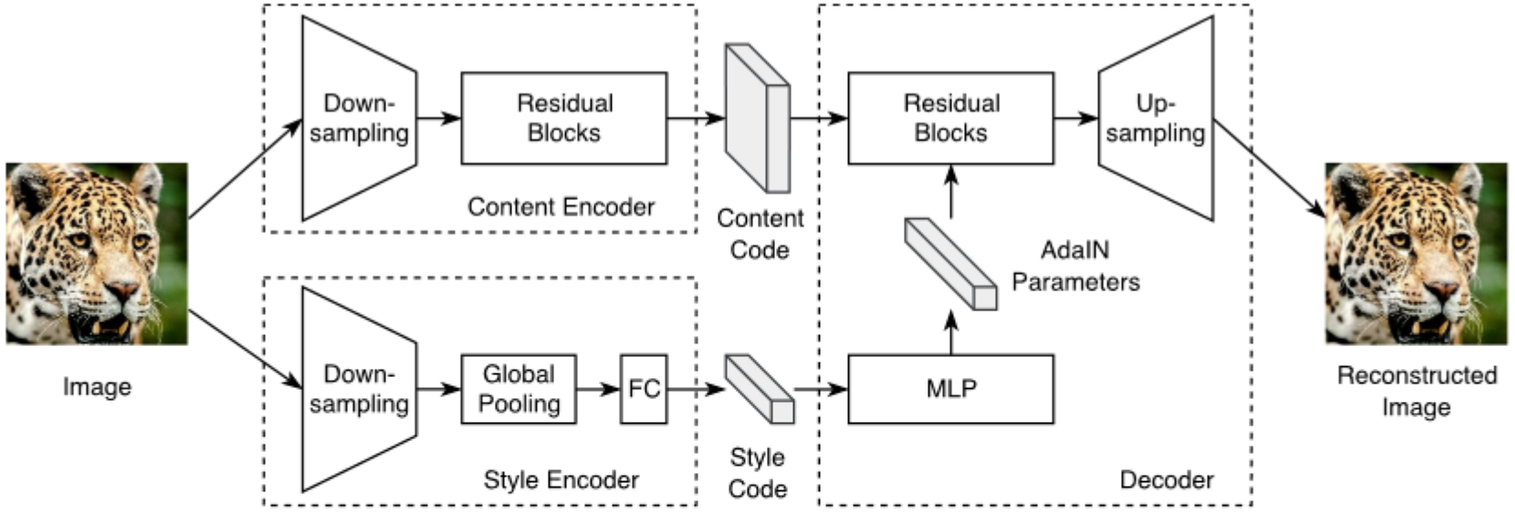
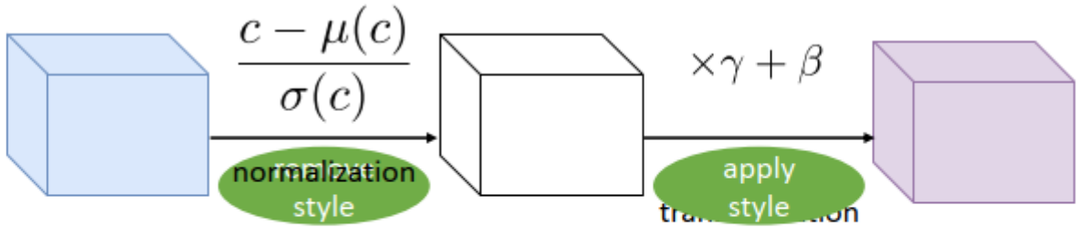


Fig. 3. Our auto-encoder architecture. The content encoder consists of several strided convolutional layers followed by residual blocks. The style encoder contains several strided convolutional layers followed by a global average pooling layer and a fully connected layer. The decoder uses a MLP to produce a set of AdaIN [54] parameters from the style code. The content code is then processed by residual blocks with AdaIN layers, and finally decoded to the image space by upsampling and convolutional layers.

Instance Normalization

Content feature: c

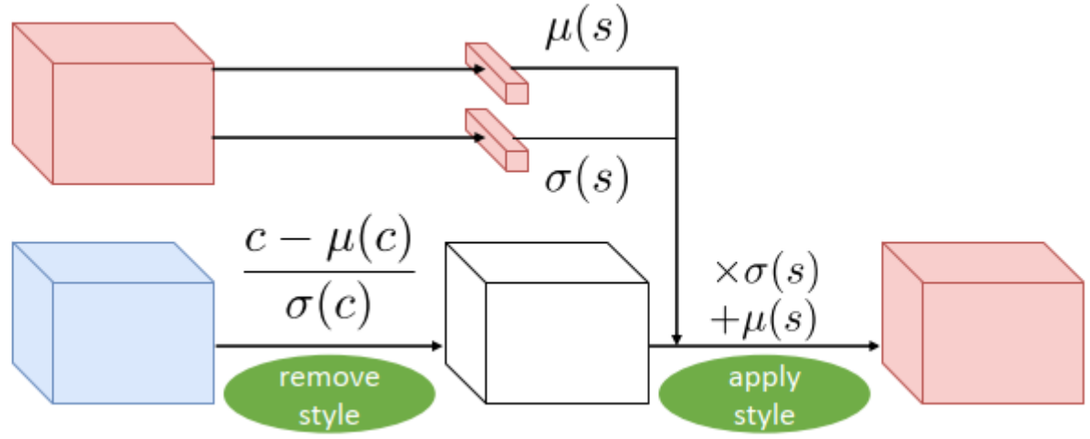


$$\mathrm{IN}(c)=\gamma\left(\frac{c-\mu(c)}{\sigma(c)}\right)+\beta$$

стиль передаём параметрами «разнормировки»

Style feature: s

Content feature: c



потом ещё к этому вернёмся

MUNIT: результаты

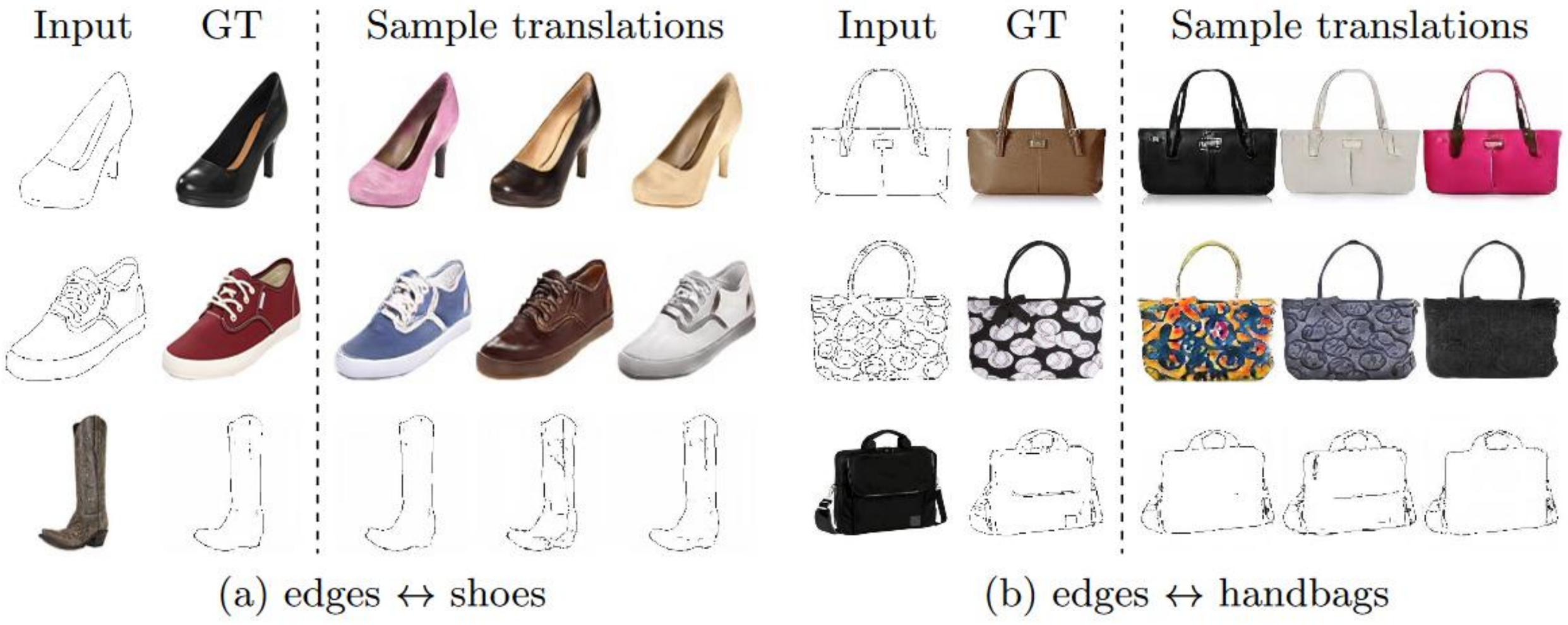


Fig. 5. Example results of (a) edges ↔ shoes and (b) edges ↔ handbags.

MUNIT: результаты

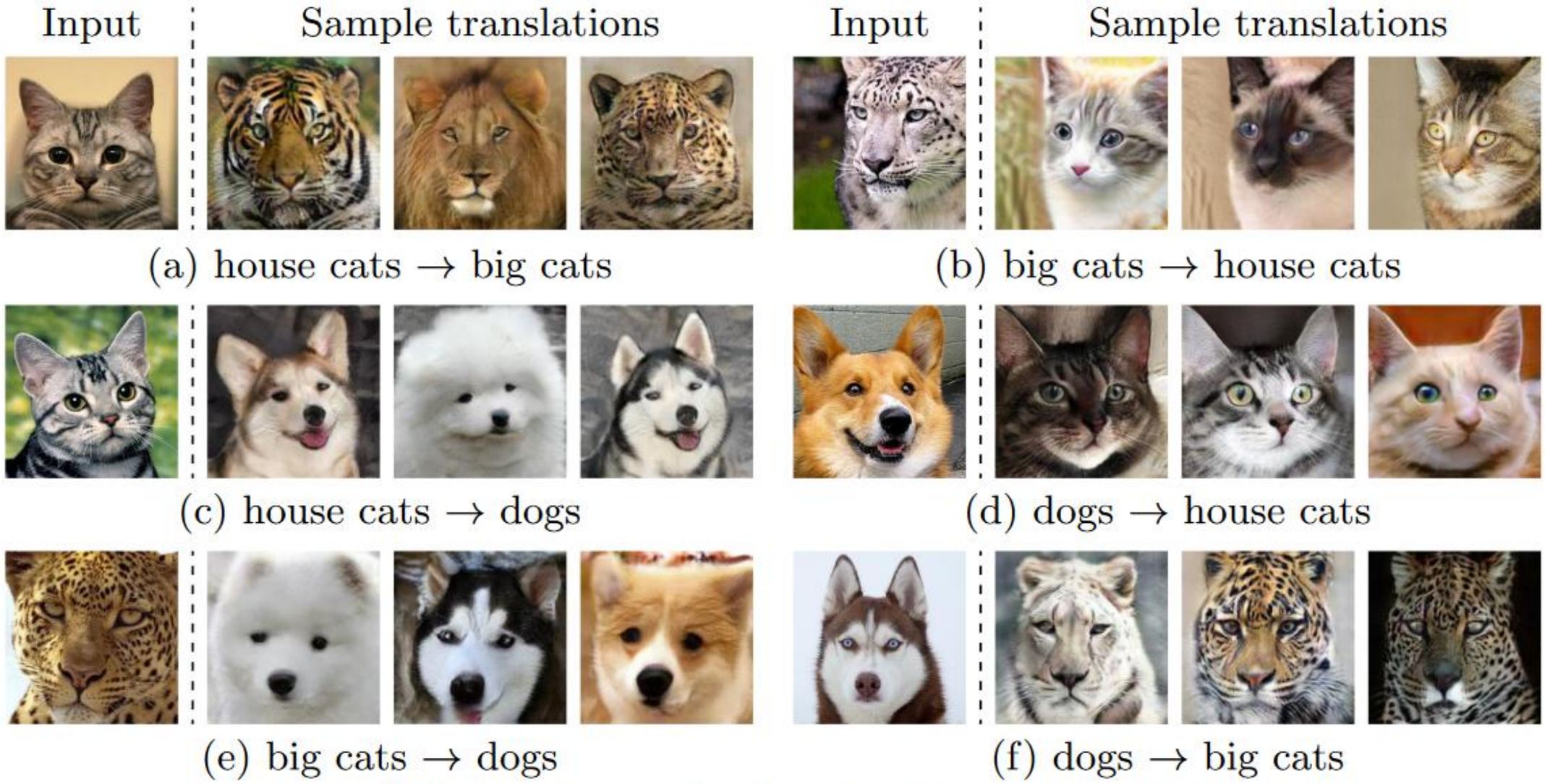
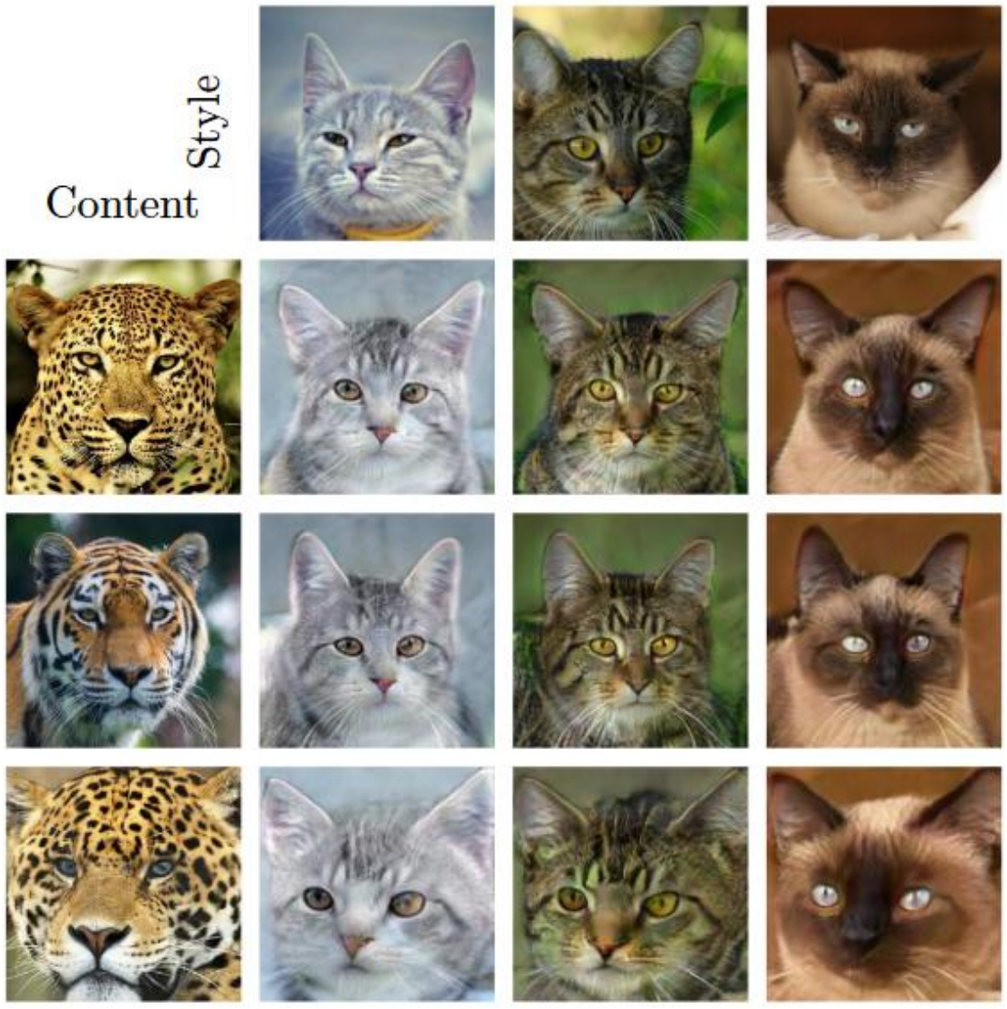


Fig. 6. Example results of animal image translation.

(a) edges → shoes



(b) big cats → house cats

Fig. 9. image translation. Each row has the same content while each column has the same style. The color of the generated shoes and the appearance of the generated cats can be specified by providing example style images.

FUNIT (NVIDIA, 2019)

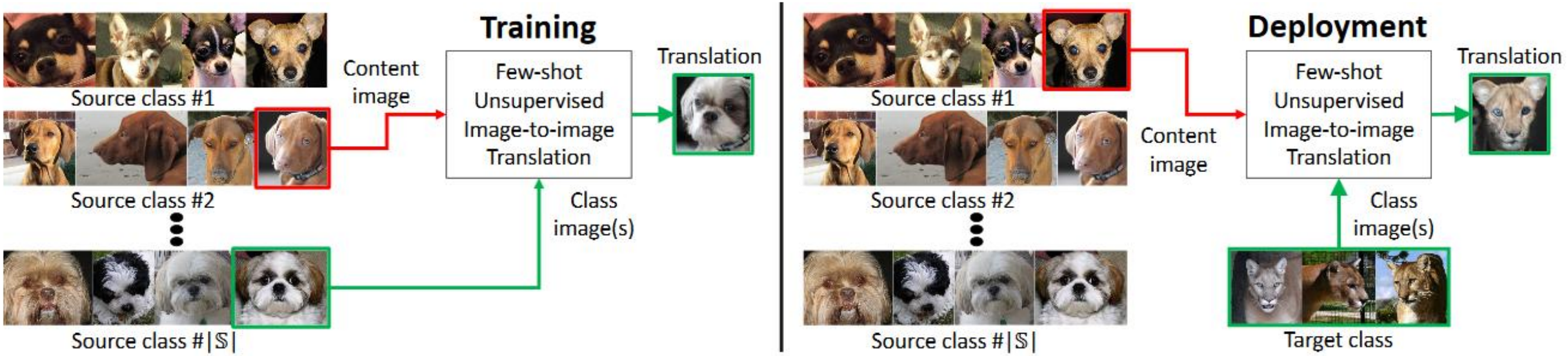


Figure 1. **Training.** The training set consists of images of various object classes (source classes). We train a model to translate images between these source object classes. **Deployment.** We show our trained model very few images of the target class, which is sufficient to translate images of source classes to analogous images of the target class even though the model has never seen a single image from the target class during training. Note that the FUNIT generator takes two inputs: 1) a content image and 2) a set of target class images. It aims to generate a translation of the input image that resembles images of the target class.

Ming-Yu Liu et al «Few-Shot Unsupervised Image-to-Image Translation» // ICCV 2019, https://nvlabs.github.io/FUNIT/

FUNIT (NVIDIA, 2019)

изображение (x) → в целевой класс
по нескольким представителям этого класса (y_1,y_2,...,y_K)

$$\bar{\mathbf{x}} = G(\mathbf{x}, \{\mathbf{y}_1, ..., \mathbf{y}_K\})$$

нет выборки пар!

- **conditional image generator G**
  - **content encoder E_x**
  - **class encoder E_y**
  - **decoder F_x**
    - **adaptive instance normalization (AdaIN) residual blocks**
    - **upscale convolutional layers**
- **multi-task adversarial discriminator D**
  для каждого класса «реально» или «трансляция»

GAN loss + content image reconstruction loss + feature matching loss

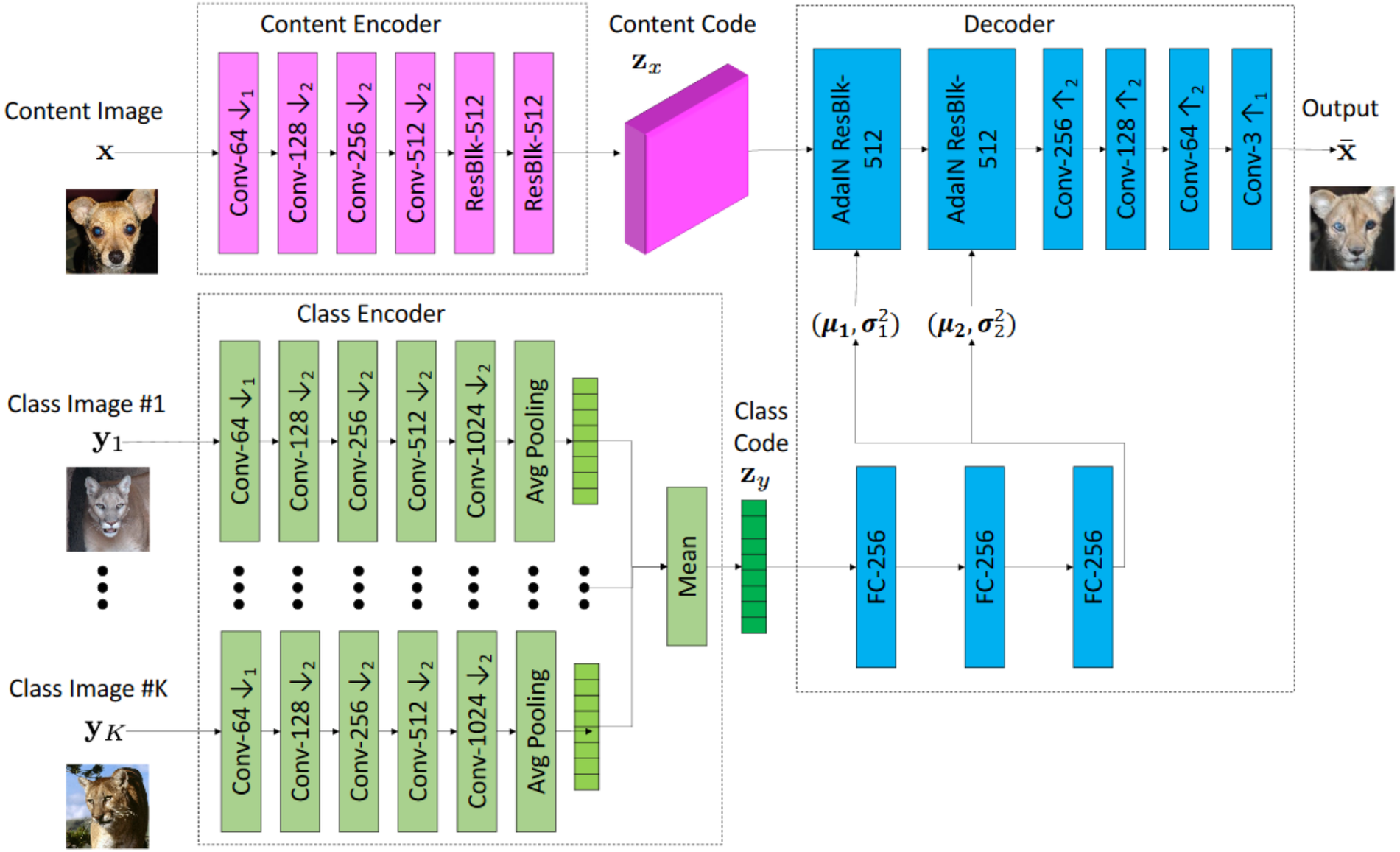


Figure 6. Visualization of the generator architecture. To generate a translation output $\bar{\boldsymbol{x}}$, the translator combines the class latent code $\boldsymbol{z}_y$ extracted from the class images $\boldsymbol{y}_1, ...\boldsymbol{y}_K$ with the content latent code $\boldsymbol{z}_x$ extracted from the input content image. Note that nonlinearity and normalization operations are not included in the visualization.


y_1
y_2
x
$\bar{x}$
y_1
y_2
x
$\bar{x}$

Немного про перенос стиля: AdaIN

Стиль кодируется в «пространственно инвариантных статистиках», например, в средних / разбросах по каналам

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

$$\mu_c(x) = \frac{1}{NHW}\sum_{n=1}^{N}\sum_{h=1}^{H}\sum_{w=1}^{W} x_{nchw}$$

$$\sigma_c(x) = \sqrt{\frac{1}{NHW}\sum_{n=1}^{N}\sum_{h=1}^{H}\sum_{w=1}^{W}(x_{nchw}-\mu_c(x))^2+\epsilon}$$

γ, β – **различны по каналам**

Instance Normalization (IN) – аналогично, но когда батч из одного объекта

D. Ulyanov, V. Lebedev, A. Vedaldi, and V. Lempitsky. Tex-ture networks: Feed-forward synthesis of textures and styl-ized images. In ICML, 2016

Adaptive Instance Normalization (AdaIN)

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

Немного про перенос стиля: AdaIN

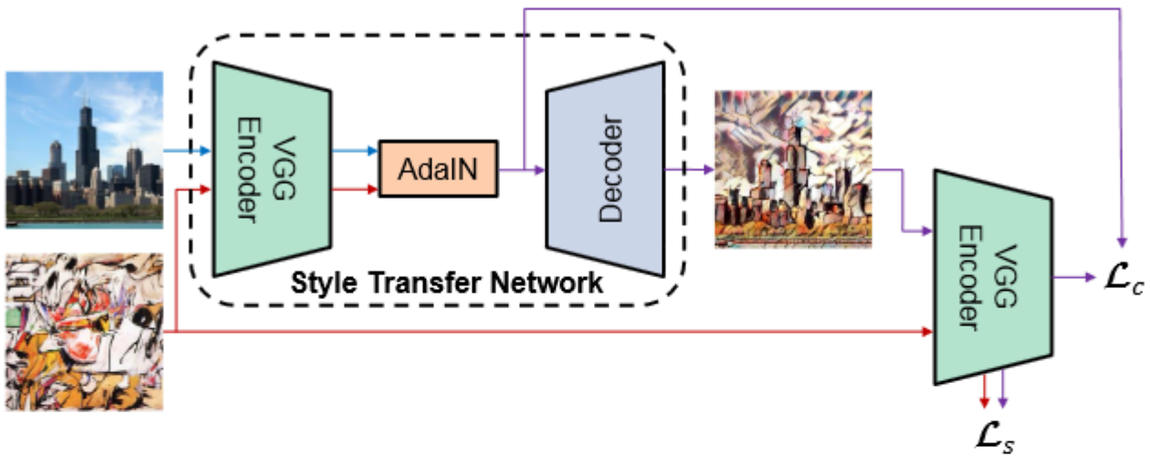
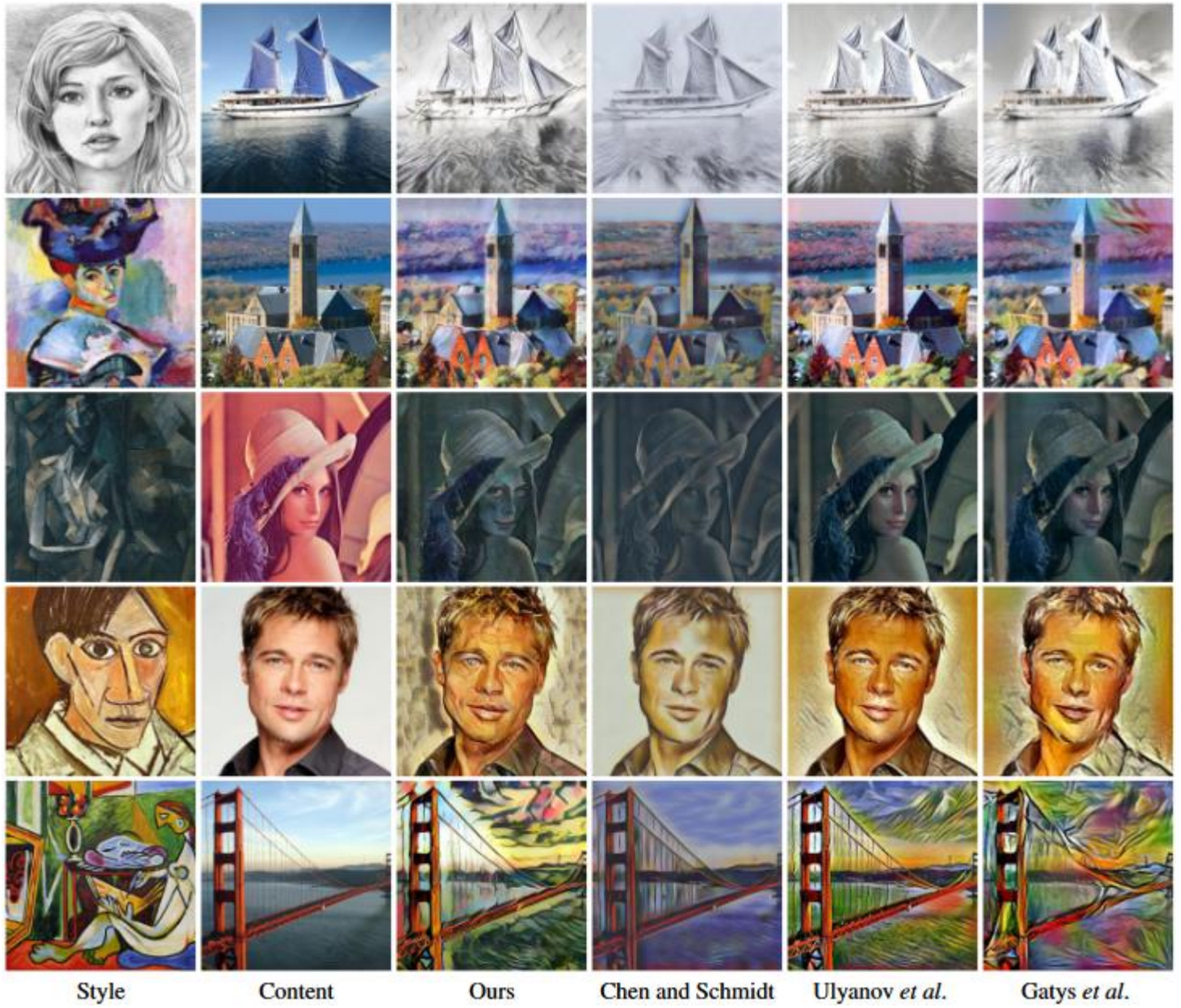
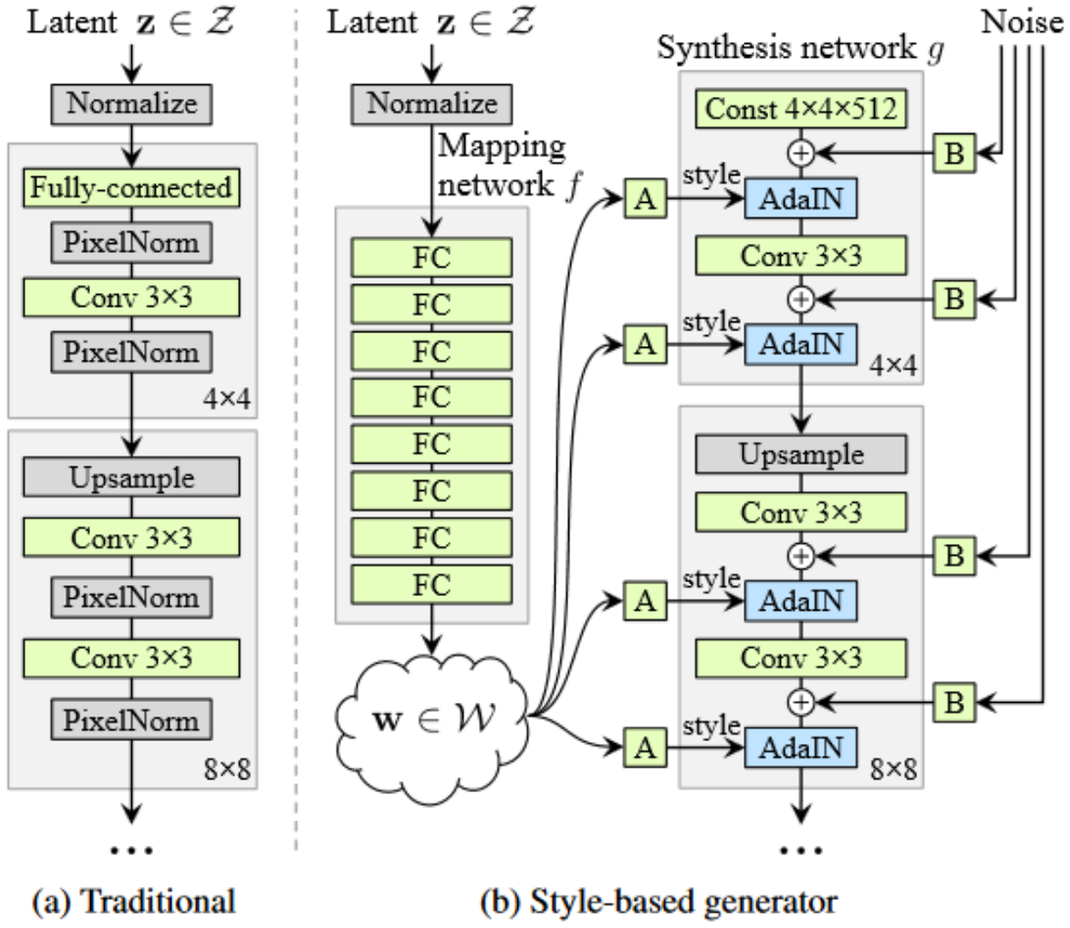


Figure 2. An overview of our style transfer algorithm. We use the first few layers of a fixed VGG-19 network to encode the content and style images. An AdaIN layer is used to perform style transfer in the feature space. A decoder is learned to invert the AdaIN output to the image spaces. We use the same VGG encoder to compute a content loss $\mathcal{L}_c$ (Equ. 12) and a style loss $\mathcal{L}_s$ (Equ. 13).



Xun Huang, Serge Belongie «Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization»
https://arxiv.org/abs/1703.06868

StyleGAN – перенос стиля



(a) Traditional (b) Style-based generator

Figure 1. While a traditional generator [30] feeds the latent code though the input layer only, we first map the input to an intermediate latent space $\mathcal{W}$, which then controls the generator through adaptive instance normalization (AdaIN) at each convolution layer. Gaussian noise is added after each convolution, before evaluating the nonlinearity. Here "A" stands for a learned affine transform, and "B" applies learned per-channel scaling factors to the noise input. The mapping network f consists of 8 layers and the synthesis network g consists of 18 layers — two for each resolution ($4^2 - 1024^2$). The output of the last layer is converted to RGB using a separate 1×1 convolution, similar to Karras et al. [30]. Our generator has a total of 26.2M trainable parameters, compared to 23.1M in the traditional generator.

Tero Karras «A Style-Based Generator Architecture for Generative Adversarial Networks»
https://arxiv.org/pdf/1812.04948.pdf

StyleGAN: особенности

Baseline Progressive GAN

последовательно увеличиваем разрешение

Bilinear Sampling

вместо nearest neighbor layers в ProGANe

Mapping Network and AdaIN

перевод в т.н. распутанное представление «disentangled representation» W

(~каждая компонента отвечает за свой признак)

из которого возможна стилизация с помощью AdaIN

4x4x512 constant value input

а не случайный вектор

Addition of Noise

B – одноканальное шумовое изображение, прибавляется ко всем каналам

вариация стилей на каждом уровне абстракции

Mixing regularization (style mixing)

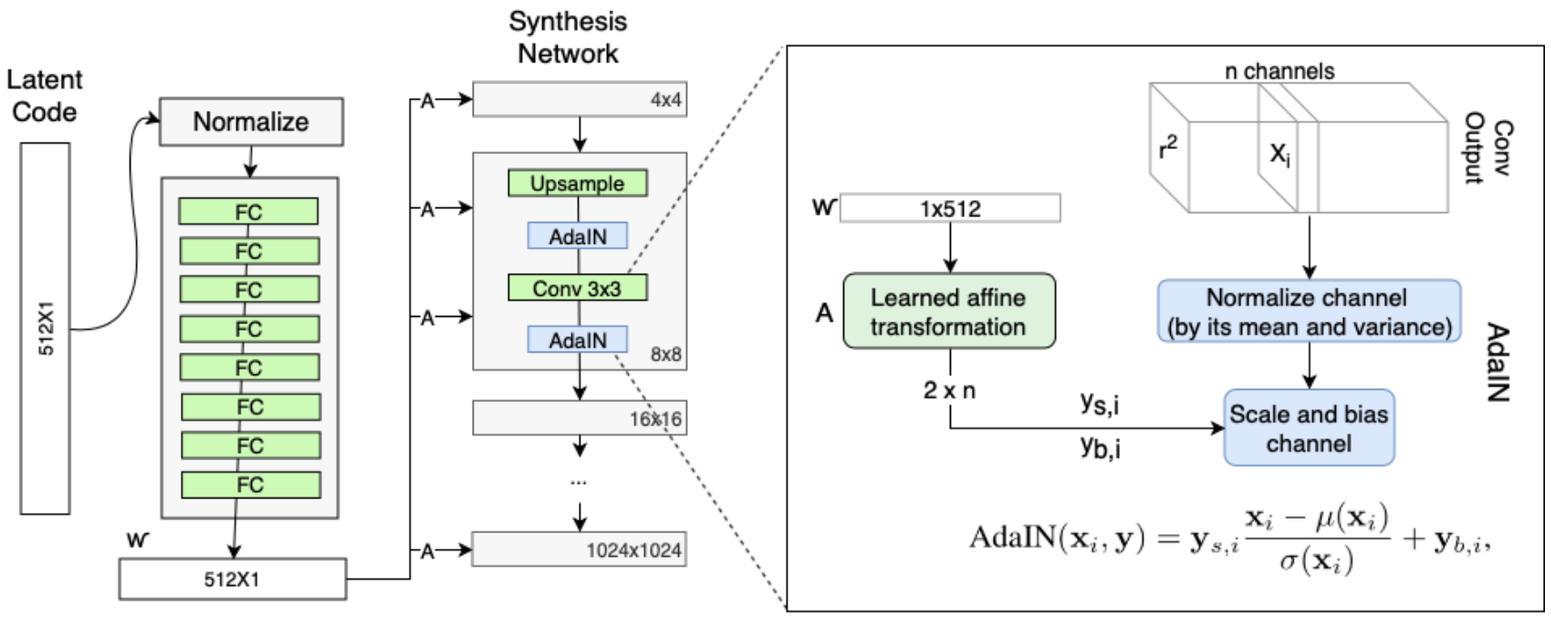
генерируется 2 стилевых вектора $z_1,z_2 \to w_1,w_2$ и точка разделения в генераторе

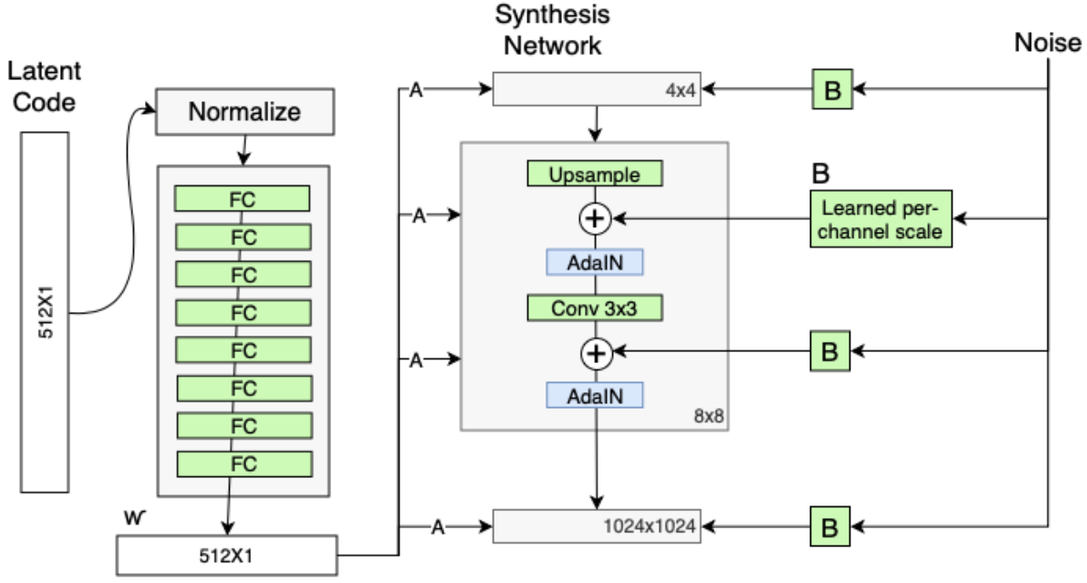
все AdaIN до точки получают один вектор, все после – второй

считается, что на каждом уровне стиль контролирует свой уровень абстракции

https://machinelearningmastery.com/introduction-to-style-generative-adversarial-network-stylegan/

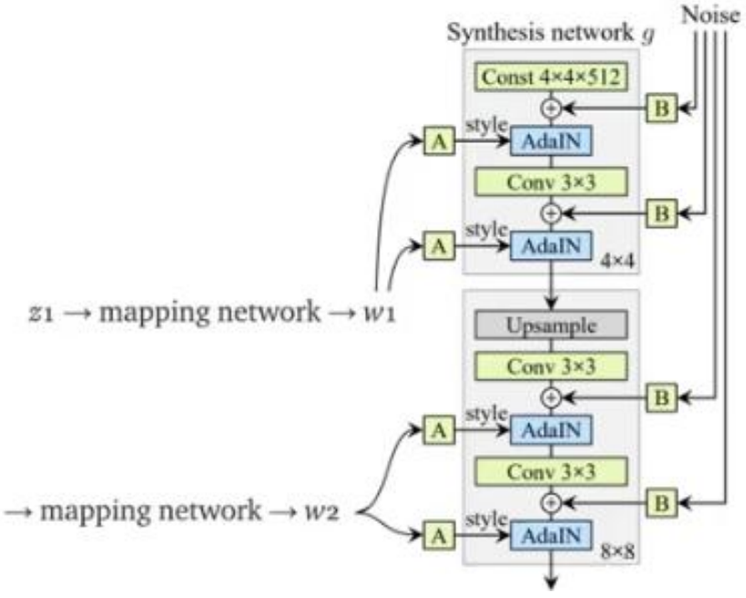
StyleGAN: подробнее





W → линейная свёртка → два конкатенированных вектора для AdaIN

обучаемая дисперсия шума по каналам



StyleGAN – перенос стиля

Figure 2. Uncurated set of images produced by our style-based generator (config F) with the FFHQ dataset. Here we used a variation of the truncation trick [42, 5, 34] with $\psi = 0.7$ for resolutions $4^2 - 32^2$. Please see the accompanying video for more results.

эксперименты на CelebA-HQ и новом датасете FFHQ

раньше: один раз подаём шум

теперь шум на каждом уровне влияет на свои абстракции:

поза / причёска / прорисовка отдельных прядей

StyleGAN – перенос стиля

Figure 3. Two sets of images were generated from their respective latent codes (sources A and B); the rest of the images were generated by copying a specified subset of styles from source B and taking the rest from source A. Copying the styles corresponding to coarse spatial resolutions ($4^2 - 8^2$) brings high-level aspects such as pose, general hair style, face shape, and eyeglasses from source B, while all colors (eyes, hair, lighting) and finer facial features resemble A. If we instead copy the styles of middle resolutions ($16^2 - 32^2$) from B, we inherit smaller scale facial features, hair style, eyes open/closed from B, while the pose, general face shape, and eyeglasses from A are preserved. Finally, copying the fine styles ($64^2 - 1024^2$) from B brings mainly the color scheme and microstructure.

смешиваем два латентных кода на разных масштабах

StyleGAN: эффект от добавления шума

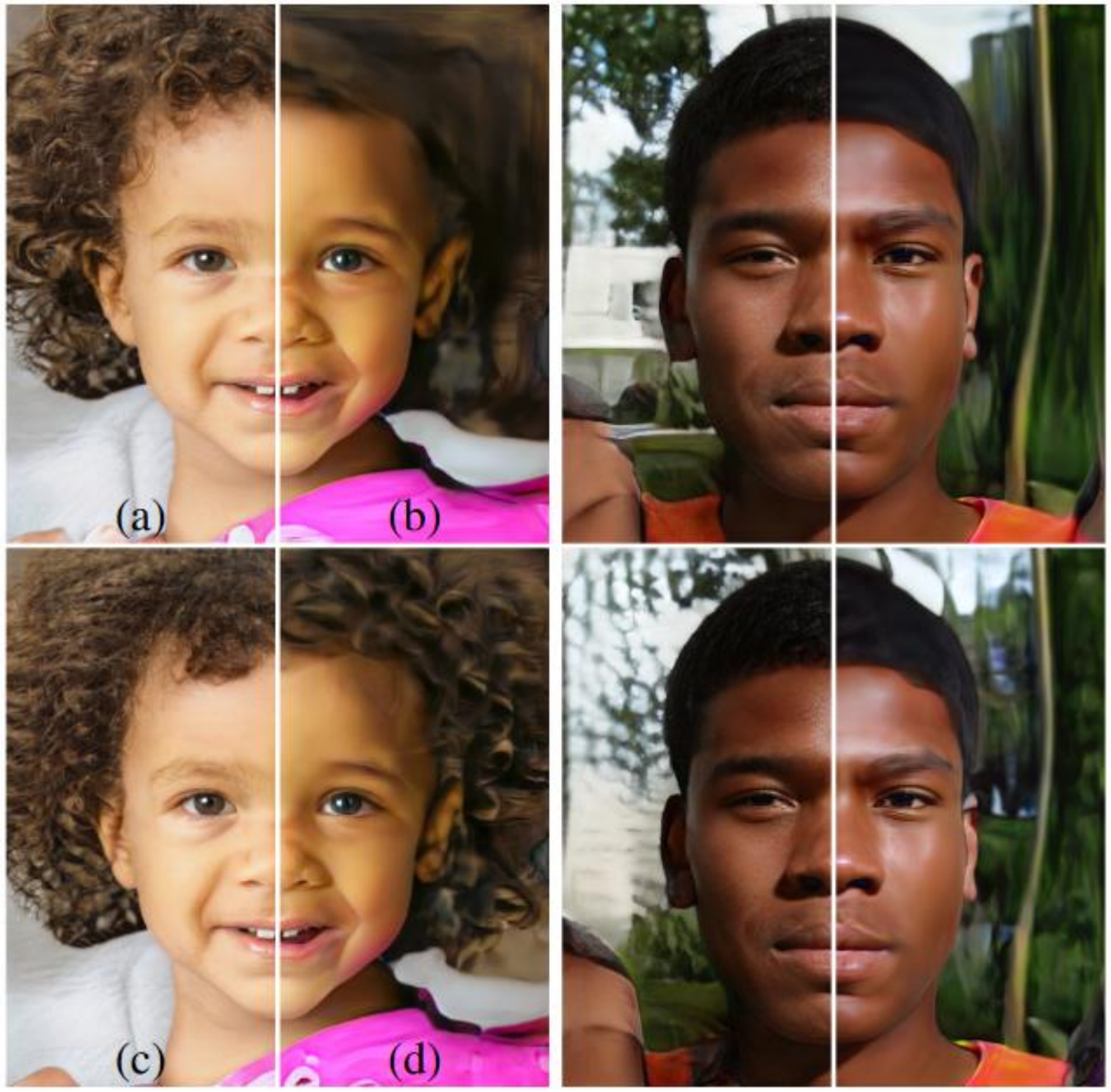


Figure 5. Effect of noise inputs at different layers of our generator. (a) Noise is applied to all layers. (b) No noise. (c) Noise in fine layers only (64^2 – 1024^2). (d) Noise in coarse layers only (4^2 – 32^2). We can see that the artificial omission of noise leads to featureless "painterly" look. Coarse noise causes large-scale curling of hair and appearance of larger background features, while the fine noise brings out the finer curls of hair, finer background detail, and skin pores.

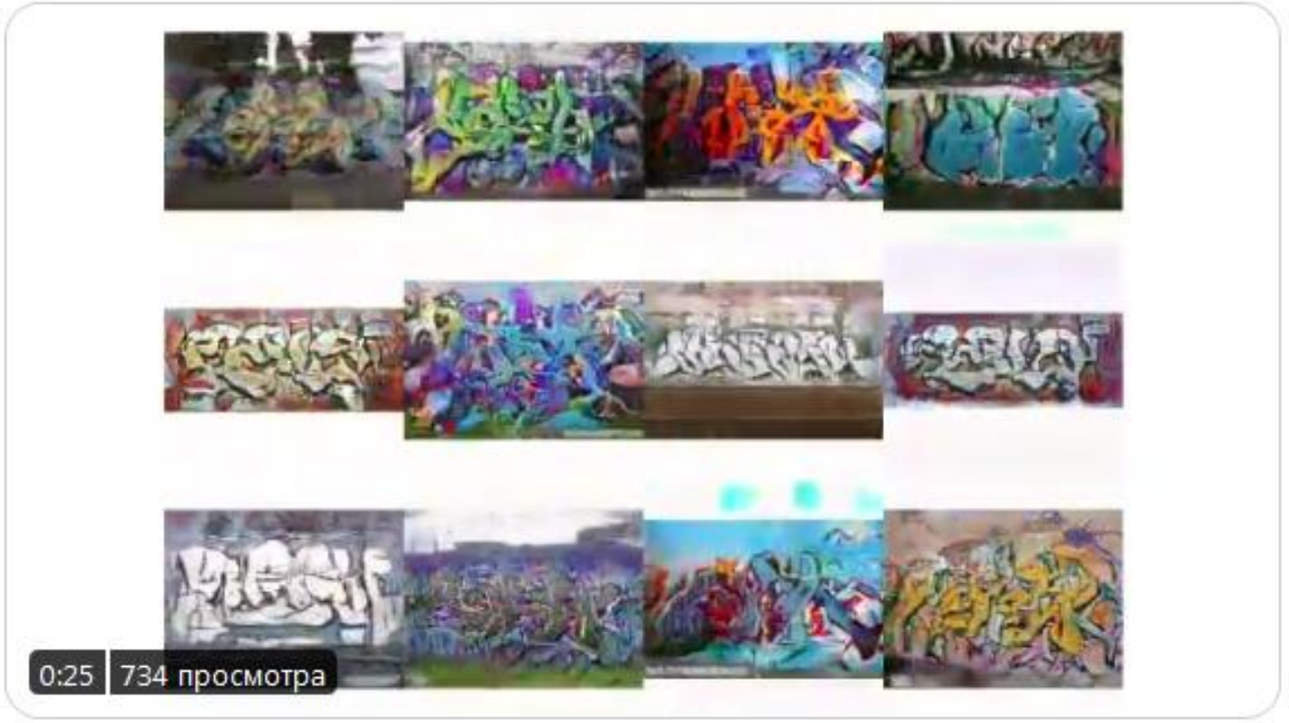
для каждого добавления шума обучается «масштаб добавления»

StyleGAN: другие объекты

Figure 10. Uncurated set of images produced by our style-based generator (config F) with the LSUN BEDROOM dataset at 256^2. FID computed for 50K images was 2.65.

Figure 11. Uncurated set of images produced by our style-based generator (config F) with the LSUN CAR dataset at 512×384. FID computed for 50K images was 3.27.

StyleGAN: шрифты / граффити



https://twitter.com/cyrildiagne/status/1095603397179396098

Есть куча применений: https://www.reg.ru/blog/anime-generation-with-stylegan/

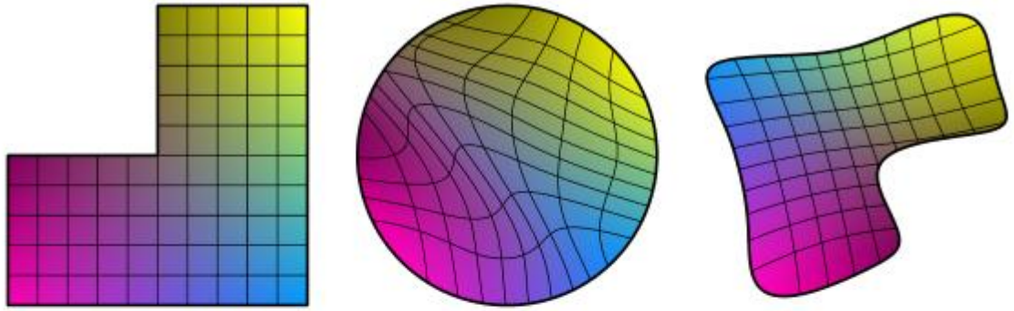
StyleGAN: затраты

Предполагаемое время обучения StyleGAN при различных разрешениях изображений и используемых GPU (источник: репозиторий StyleGAN)

Число используемых GPU	1024^2	512^2	256^2	[Апрель 2019 г., тарифы REG.RU]
1	41 день 4 часа [988 GPU-часов]	24 дня 21 час [597 GPU-часов]	14 дней 22 часа [358 GPU-часов]	[₽89к, ₽54к, ₽32к]
2	21 день 22 часа [1052]	13 дней 7 часов [638]	9 дней 5 часов [442]	[105к, 64к, 44к]
4	11 дней 8 часов [1088]	7 дней 0 часов [672]	4 дня 21 час [468]	[131к, 81к, 56к]
8	6 дней 14 часов [1264]	4 дня 10 часов [848]	3 дня 8 часов [640]	[177к,, 119к, 90к]

https://www.reg.ru/blog/anime-generation-with-stylegan/

«Disentangled representation»



(a) Distribution of features in training set

(b) Mapping from $\mathcal{Z}$ to features

(c) Mapping from $\mathcal{W}$ to features

Figure 6. Illustrative example with two factors of variation (image features, e.g., masculinity and hair length). (a) An example training set where some combination (e.g., long haired males) is missing. (b) This forces the mapping from $\mathcal{Z}$ to image features to become curved so that the forbidden combination disappears in $\mathcal{Z}$ to prevent the sampling of invalid combinations. (c) The learned mapping from $\mathcal{Z}$ to $\mathcal{W}$ is able to "undo" much of the warping.

«распутывание» – хотелось бы, чтобы компоненты случайного вектора отвечали за конкретные признаки изображения (поза, причёска, разрез глаз и т.п.)

один из вариантов требований – «ортогональность»

опускаем, какие исследования были в StyleGAN

StyleGAN2: избавление от артефактов

Figure 1. Instance normalization causes water droplet -like artifacts in StyleGAN images. These are not always obvious in the generated images, but if we look at the activations inside the generator network, the problem is always there, in all feature maps starting from the 64x64 resolution. It is a systemic problem that plagues all StyleGAN images.

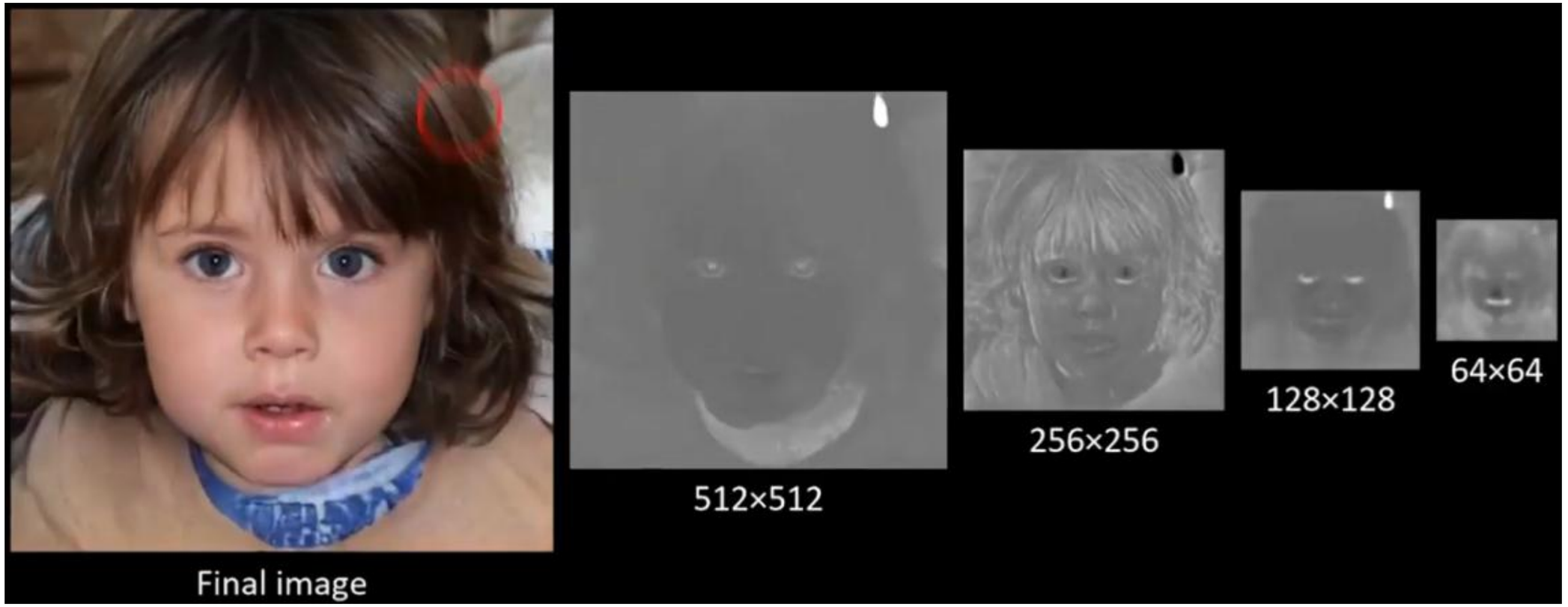
Попытка борьбы с артефактами – «Removing normalization artifacts»

убрали лишние операции в начале, прибавление смещения вынесено

AdaIN → «демодуляция»

Tero Karras, et al. «Analyzing and Improving the Image Quality of StyleGAN» // https://arxiv.org/abs/1912.04958

StyleGAN2: избавление от артефактов



артефакты похожи на «кучку»

появляются с какого-то слоя (ближе к высоким разрешениям)

демодуляция помогла их устранить

https://www.youtube.com/watch?v=MYCTn80qSk0

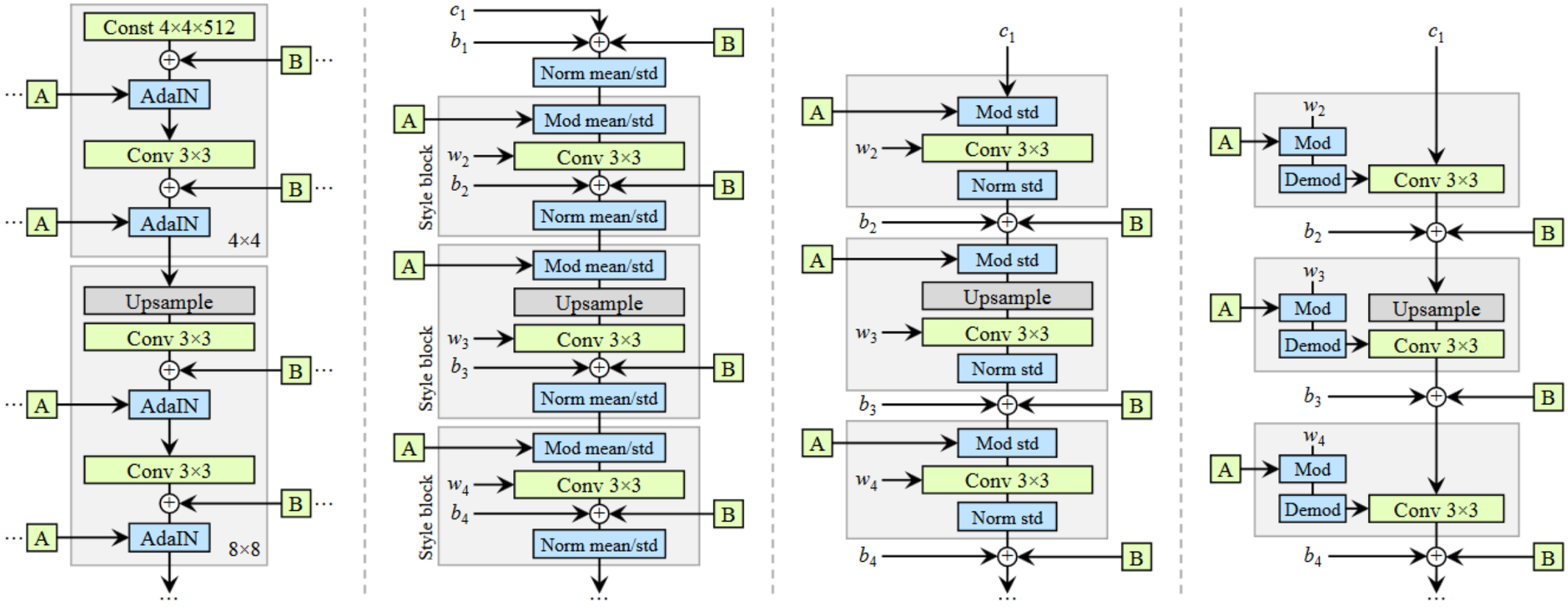


Figure 2. We redesign the architecture of the StyleGAN synthesis network. (a) The original StyleGAN, where A denotes a learned affine transform from $\mathcal{W}$ that produces a style and B is a noise broadcast operation. (b) The same diagram with full detail. Here we have broken the AdaIN to explicit normalization followed by modulation, both operating on the mean and standard deviation per feature map. We have also annotated the learned weights (w), biases (b), and constant input (c), and redrawn the gray boxes so that one style is active per box. The activation function (leaky ReLU) is always applied right after adding the bias. (c) We make several changes to the original architecture that are justified in the main text. We remove some redundant operations at the beginning, move the addition of b and B to be outside active area of a style, and adjust only the standard deviation per feature map. (d) The revised architecture enables us to replace instance normalization with a "demodulation" operation, which we apply to the weights associated with each convolution layer.

StyleGAN2: что такое демодуляция (Demod)

после модуляции (т.е. изменения масштаба без смещений)
веса свёрток как бы стали такими:

$$w_{c,i_{\text{out}},s} \to s_c \cdot w_{c,i_{\text{out}},s}$$

каждая признаковая карта получила свой масштаб
i_out – номер выходной свёртки (номер новой признаковой карты)
s – двумерный индекс H×W

демодуляция:

$$w'_{c,i_{\text{out}},s} \to \frac{w'_{c,i_{\text{out}},s}}{\sqrt{\sum_{c,s} w'^{2}_{c,i_{\text{out}},s} + \varepsilon}}$$

StyleGAN2

Проблемы «Progressive growing»

Figure 6. Progressive growing leads to "phase" artifacts. In this example the teeth do not follow the pose but stay aligned to the camera, as indicated by the blue line.

«Lazy regularization» – учитывать регуляризационное слагаемое 1 раз из 16 батчей

Пересмотр архитектуры генератора

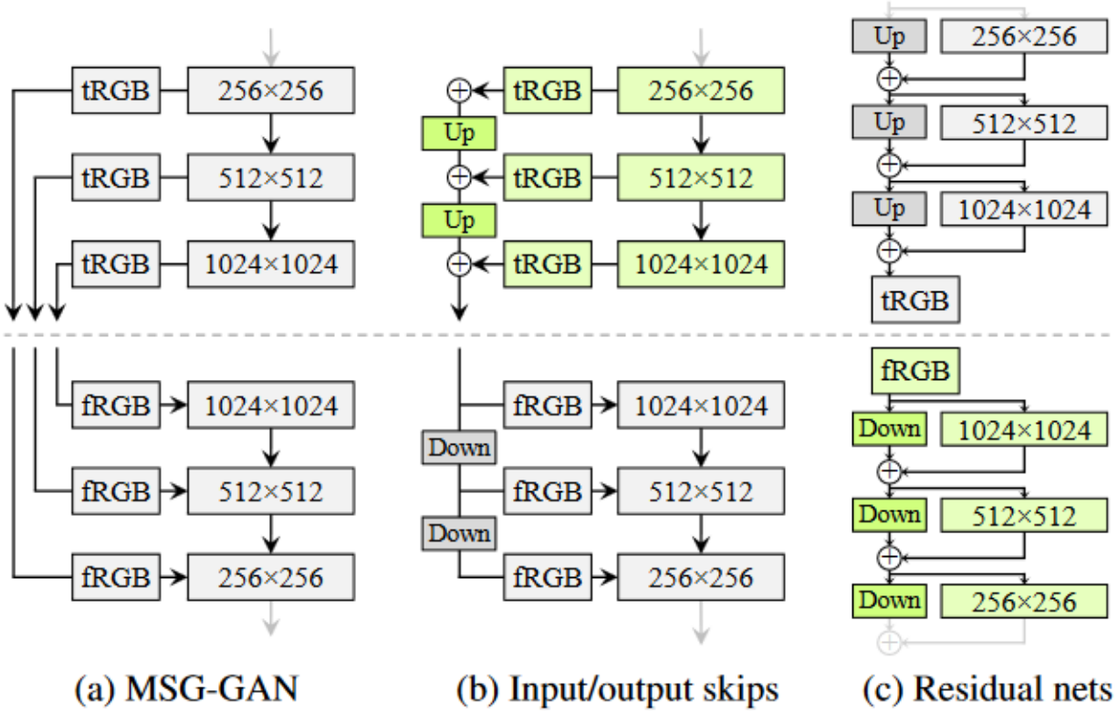


Figure 7. Three generator (above the dashed line) and discriminator architectures. Up and Down denote bilinear up and downsampling, respectively. In residual networks these also include 1×1 convolutions to adjust the number of feature maps. tRGB and fRGB convert between RGB and high-dimensional per-pixel data. Architectures used in configs E and F are shown in green.

StyleGAN2

Configuration	FFHQ, 1024×1024				LSUN Car, 512×384			
	FID ↓	Path length ↓	Precision ↑	Recall ↑	FID ↓	Path length ↓	Precision ↑	Recall ↑
A Baseline StyleGAN [24]	4.40	212.1	**0.721**	0.399	3.27	1484.5	**0.701**	0.435
B + Weight demodulation	4.39	175.4	0.702	0.425	3.04	862.4	0.685	0.488
C + Lazy regularization	4.38	158.0	0.719	0.427	2.83	981.6	0.688	0.493
D + Path length regularization	4.34	**122.5**	0.715	0.418	3.43	651.2	0.697	0.452
E + No growing, new G & D arch.	3.31	124.5	0.705	0.449	3.19	471.2	0.690	0.454
F + Large networks (StyleGAN2)	**2.84**	145.0	0.689	**0.492**	**2.32**	**415.5**	0.678	**0.514**
Config A with large networks	3.98	199.2	0.716	0.422	–	–	–	–

Table 1. Main results. For each training run, we selected the training snapshot with the lowest FID. We computed each metric 10 times with different random seeds and report their average. *Path length* corresponds to the PPL metric, computed based on path endpoints in $\mathcal{W}$ [24], without the central crop used by Karras et al. [24]. The FFHQ dataset contains 70k images, and the discriminator saw 25M images during training. For LSUN CAR the numbers were 893k and 57M. ↑ indicates that higher is better, and ↓ that lower is better.

Dataset	Resolution	StyleGAN (A)		StyleGAN2 (F)	
		FID	PPL	FID	PPL
LSUN CAR	512×384	3.27	1485	**2.32**	**416**
LSUN CAT	256×256	8.53	924	**6.93**	**439**
LSUN CHURCH	256×256	4.21	742	**3.86**	**342**
LSUN HORSE	256×256	3.83	1405	**3.43**	**338**

Table 3. Improvement in LSUN datasets measured using FID and PPL. We trained CAR for 57M images, CAT for 88M, CHURCH for 48M, and HORSE for 100M images.

Итоги

Авторегрессионные модели

– нет интерполяций (между двумя изображениями)

– пока очень медленные

– нет хорошего латентного представления

+ хороши в видео, аудио, картинках, текстах

+ явное задание плотности / лучшее правдоподобие

– неэффективная последовательная генерация

VAE

– оптимизация ELBO

+ полезное представление через скрытые переменные, по x находим z

+ «сжатое представление» («compressed» representation learning)

- интерполяции

– мутные изображения

– возможны неадекватные ограничения на латентное пр-во

– пока не на очень больших картинках

Потоки

+ можно применять совместно с другими методами

+ красивая математика

+ обратимое преобразование

+ легко сэмплировать, оценивать p(x)

– латентное пр-во огромно (такое же, как исходное)

GAN итоги

+ игровой подход

– в обучении нет MLE

– в явном виде не вычисляют P(X), сложно найти z

т.е. в чистом виде только для сэмплирования и генерации

+ устойчивость к переобучению

(генератор не видит данных)

+ сейчас – лучшие / быстрые / чёткие картинки

+ хорошие условные модели

– есть трюки в обучении

– нет теоретического понимания

– однообразность (Mode collapse)

Выученные признаки можно использовать в SL!

TABLE 1: Comparison between deep generative models in terms of training and test speed, parameter efficiency, sample quality, sample diversity, and ability to scale to high resolution data. Quantitative evaluation is reported on the CIFAR-10 dataset [114] in terms of Fréchet Inception Distance (FID) and negative log-likelihood (NLL) in bits-per-dimension (BPD).

Method	Train Speed	Sample Speed	Param. Effic.	Sample Quality	Relative Divers.	Resolution Scaling	FID	NLL (in BPD)
Generative Adversarial Networks								
DCGAN [169]	★★★★★	★★★★★	★★★★☆	★★★☆☆	★☆☆☆☆	★★☆☆☆	17.70	-
ProGAN [102]	★★☆☆☆	★★★★★	★★★★☆	★★★☆☆	★★☆☆☆	★★★★★	15.52	-
BigGAN [17]	★★★★☆	★★★★★	★★★★☆	★★★★☆	★☆☆☆☆	★★★★☆	14.73	-
StyleGAN2 + ADA [103]	★★★☆☆	★★★★★	★★★★☆	★★★★☆	★★☆☆☆	★★★★★	2.42	-
Energy Based Models								
IGEBM [42]	★★★☆☆	★★☆☆☆	★★★★★	★★★☆☆	★★★☆☆	★★★☆☆	37.9	-
Denoising Diffusion [80]	★★☆☆☆	★★☆☆☆	★★★☆☆	★★★★☆	★★★★☆	★★★★☆	3.17	≤ 3.75
DDPM++ Continuous [191]	★★☆☆☆	★★★☆☆	★★★☆☆	★★★★☆	★★★★☆	★★★★☆	2.92	2.99
Flow Contrastive [51]	★☆☆☆☆	★★★☆☆	★☆☆☆☆	★★☆☆☆	★★★☆☆	★★☆☆☆	37.30	≈ 3.27
VAEBM [226]	★★☆☆☆	★★★☆☆	★★★☆☆	★★★☆☆	★★★☆☆	★★★★☆	12.19	-
Variational Autoencoders								
Convolutional VAE [110]	★★★★★	★★★★★	★★★★☆	★★☆☆☆	★★★★☆	★★☆☆☆	106.37	≤ 4.54
Variational Lossy AE [27]	★☆☆☆☆	★☆☆☆☆	★☆☆☆☆	★★☆☆☆	★★★★☆	★☆☆☆☆	-	≤ 2.95
VQ-VAE [171], [215]	★★☆☆☆	★★★☆☆	★★☆☆☆	★★★☆☆	★★★★☆	★★★★★	-	≤ 4.67
VD-VAE [29]	★★★☆☆	★★★★☆	★★★☆☆	★★★☆☆	★★★★☆	★★★★★	-	≤ 2.87
Autoregressive Models								
PixelRNN [214]	★☆☆☆☆	★☆☆☆☆	★☆☆☆☆	★☆☆☆☆	★★★★★	★★★☆☆	-	3.00
Gated PixelCNN [213]	★☆☆☆☆	★☆☆☆☆	★☆☆☆☆	★★☆☆☆	★★★★★	★★★☆☆	65.93	3.03
PixelIQN [161]	★☆☆☆☆	★☆☆☆☆	★☆☆☆☆	★★☆☆☆	★★★★★	★★★☆☆	49.46	-
Sparse Trans. + DistAug [30], [99]	★★☆☆☆	★☆☆☆☆	★★☆☆☆	★★★★☆	★★★★★	★★★☆☆	14.74	2.66
Normalizing Flows								
RealNVP [39]	★☆☆☆☆	★★★★☆	★☆☆☆☆	★☆☆☆☆	★★★★★	★★★☆☆	-	3.49
Masked Autoregressive Flow [165]	★★☆☆☆	★☆☆☆☆	★★☆☆☆	★☆☆☆☆	★★★☆☆	★★☆☆☆	-	4.30
GLOW [111]	★☆☆☆☆	★★★★☆	★☆☆☆☆	★★☆☆☆	★★★★★	★★★★☆	45.99	3.35
FFJORD [56]	★☆☆☆☆	★★★☆☆	★★★★★	★★☆☆☆	★★★★★	★★☆☆☆	-	3.40
Residual Flow [24]	★★☆☆☆	★★★★☆	★★★★☆	★★☆☆☆	★★★★★	★★★★☆	46.37	3.28

Ссылки

http://whichfaceisreal.com – **демонстрация StyleGANa**

https://thispersondoesnotexist.com/ – **демонстрация StyleGANa**

https://youtu.be/c-NJtV9Jvp0 – **демонстрация StyleGANa2**

неплохой, но немного старый обзор
https://jonathan-hui.medium.com/gan-gan-series-2d279f906e7b

обзор генеративных моделей
Sam Bond-Taylor, Adam Leach, Yang Long, Chris G. Willcocks «Deep Generative Modelling: A Comparative Review of VAEs, GANs, Normalizing Flows, Energy-Based and Autoregressive Models» https://arxiv.org/pdf/2103.04922.pdf