**Оглавление**

[**Введение** 1](#_Toc196998632)

[**Основы обучения GAN-ов** 2](#_Toc196998633)

[**Метрики качества** 7](#_Toc196998634)

[**Базовые модели** 10](#_Toc196998635)

[**Условная генерация** 11](#_Toc196998636)

[**StyleGAN** 13](#_Toc196998637)

[**StyleGAN-2** 14](#_Toc196998638)

[**StyleGAN-ADA** 15](#_Toc196998639)

[**Применения генеративных состязательных нейросетей** 16](#_Toc196998640)

[**Inpainting** 16](#_Toc196998641)

[**Outpainting** 17](#_Toc196998642)

[**Редактирование изображений** 17](#_Toc196998643)

**Генеративно-состязательные сети (GAN)**

# **Введение**

Генеративно-состязательные сети (Generative Adversarial Networks, GAN) – это большой класс генеративных моделей, общая черта которых заключается в том, что они обучаются одновременно с другой сетью, которая старается отличить сгенерированные объекты от настоящих.

Изображение выглядит как текст, черно-белый, Шрифт, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

# **Основы обучения GAN-ов**

Изображение выглядит как текст, снимок экрана, мультфильм

Контент, сгенерированный ИИ, может содержать ошибки.

Классическая аналогия того, как учатся GANы — это фальшивомонетчик и полицейский. Задача фальшивомонетчика — научиться создавать купюры, которые полицейский не сможет отличить от реальных. Задача полицейского — научиться отличать купюры фальшивомонетчика от настоящих.

Чтобы понять, как обучаются GANы, надо представить себе следующий мысленный эксперимент. Допустим, фальшивомонетчик и полицейский — друзья, которые решили поучиться друг у друга. Фальшивомонетчик создаёт несколько фальшивых купюр и показывает полицейскому. Полицейский говорит фальшивомонетчику, какие из его купюр, по его мнению, поддельные, а какие — настоящие. Фальшивомонетчик запоминает отзыв полицейского и в следующий раз улучшит свои купюры на основе отзыва от полицейского. Сам полицейский при этом тоже учится: он запоминает, что купюры, которые он видел — поддельные.

В нашем мысленном эксперименте представим, что фальшивомонетчик взаимодействует с полицейским много раз. Что получается в результате? С каждым разом купюры фальшивомонетчика всё труднее отличить от настоящих. И с каждым разом умение выявлять поддельные купюры у полицейского выше.

Важный вопрос для понимания работы GANов: в какой момент мы можем утверждать, что фальшивомонетчик хорошо подделывает купюры?

Ответ:  
Когда фальшивомонетчик сможет обманывать сильного полицейского. В начале нашего эксперимента полицейский плохо отличает подделку от оригинала. Поэтому обмануть его можно купюрами плохого качества. Нам же интересно получить фальшивомонетчика, который будет выдавать купюры, неотличимые от оригинала даже профессионалом.

# **Метрики качества**

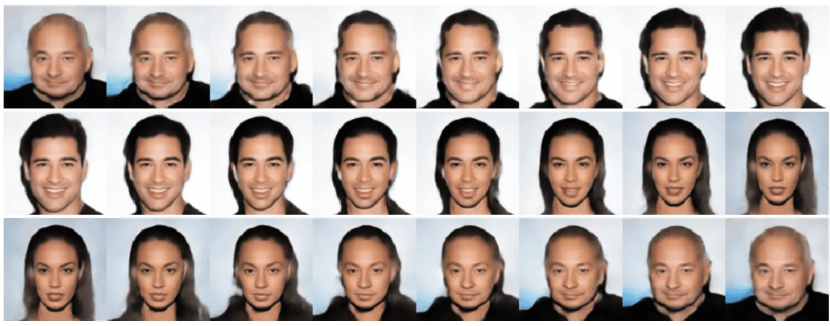
После успешного обучения генератора хотелось бы также понять, насколько хорошо он работает. Для этого рассмотрим на примере задачи генерации изображений типовые ошибки, которые может совершать GAN. Наиболее частая проблема – плохое качество или наличие артефактов – вызвана ограничениями, связанными с возможностями генератора и несовершенством самих методов обучения. Здесь всё просто: наша генеративная модель плохо работает, и мы это видим на сгенерированных сэмплах. Более скрытым видом ошибок является так называемый mode collapse: обученный генератор выдаёт реалистично выглядящие картинки, но они не покрывают всё разнообразие распределения *p*(*x*). Например, если наша модель учится генерировать изображения с животными, то она может проигнорировать более редкие виды, а научиться генерировать только наиболее часто встречающиеся. Более экстремальная форма подобного поведения – это когда модель вообще выдаёт вариацию одной картинки. Иногда в литературе общее качество результатов работы нейросети, по аналогии с задачей классификации, измеряется точностью метода (precision), а отсутствие mode collapse измеряется полнотой (recall).

Самый простой и действенный способ измерить как precision, так и recall – сгенерировать данные и посмотреть на них, дав экспертную оценку уровня их реализма.

**Интерполяции в скрытом пространстве**

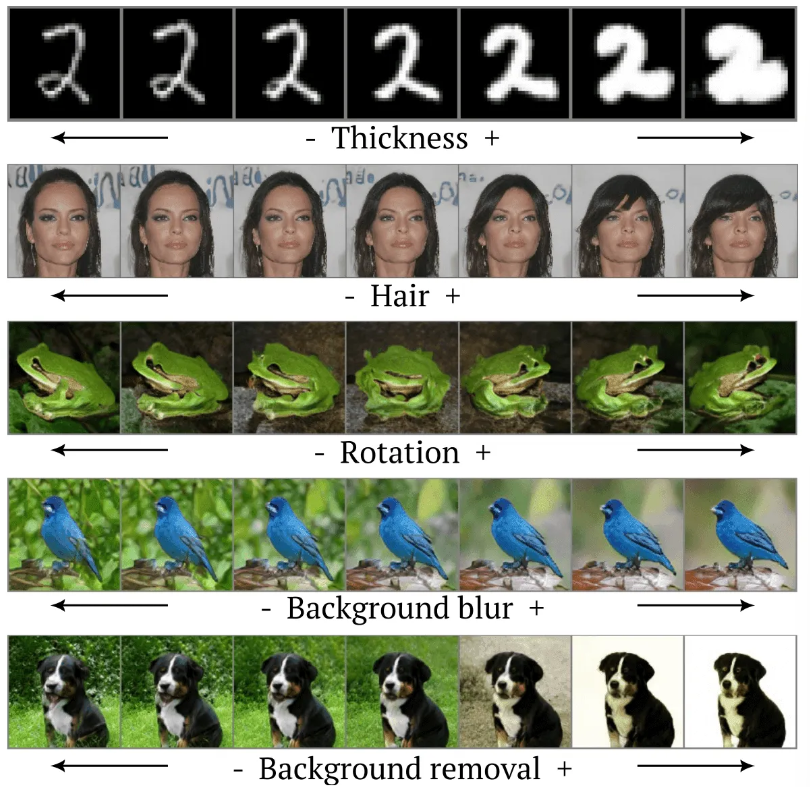
Ещё один способ измерения качества, который мы рассмотрим, напрямую связан с тем, что генеративно-состязательные модели эффективно занимаются кодированием потенциально высокоразмерных данных в низкоразмерное представление. Но как для нейросети с большим числом параметров проверить, занимается ли она реальным кодированием или простым запоминанием выборки?

Одним из тестов на подобное поведение является интерполяция между сгенерированными примерами. Возьмём два случайных вектора ​ и  из *p*(*z*). Рассмотрим все векторы, которые лежат между ними z=α+(1−α), α∈[0,1]. К каждому такому вектору *z* применим наш генератор и получим  для промежуточных векторов и ​ для ​. Для правильно обученного GANа мы должны увидеть следующую картинку: при изменении коэффициента *α* изображение  должно плавно меняться и перетекать из  в ​. При этом каждая промежуточная картинка должна быть так же реалистичным сэмплом.



Качество такой интерполяции сложно измерить численно, но если мы видим, что промежуточные результаты меняются случайно без какой-либо связи с семантикой интерполируемых примеров, то это говорит о плохом качестве генератора.

Интерполяция в скрытом пространстве с недавних пор стала использоваться для генерации анимаций и видео. Ведь анимация — это последовательность кадров, плавно переходящих друг в друга. И если у нас есть обученный GAN для генерации картинок, то нам нужно лишь найти путь в скрытом пространстве таким образом, чтобы набор сгенерированных картинок складывался в анимацию. Более того, в скрытом пространстве можно находить различные интерпретируемые пути. Например, путь, при движении по которому размывается задний фон или меняется причёска.



# **Базовые модели**

Чтобы лучше понимать современные модели, давайте сначала рассмотрим более базовые модели. Хотя они редко используются напрямую, многие идеи из них легли в основу современных моделей.

**DCGAN**

Наиболее простая версия генеративной модели для изображений — это DCGAN ([Deep Convolutional GAN](https://arxiv.org/pdf/1511.06434.pdf), 2015 год). Её до сих пор можно иногда встретить как в литературе, так и на практике.

В основе DCGAN лежит простая идея: нейросети, основанные на свёртках, отлично подходят для распознавания изображений, а значит вполне могут подойти и для их генерации. Единственное отличие, которое требуется – это постепенно увеличивать внутри нейросети пространственный размер признаков, а не уменьшать. Для этого в современных нейросетях делается операция nearest upsampling, очень похожая на max pooling. В nearest upsampling пространственное разрешение карты признаков увеличивается за счёт того, что каждый вектор повторяется *K* раз по горизонтали и по вертикали. К примеру, после увеличения таким образом карты признаков, состоящей из одной единицы, мы получим квадрат размера *K*×*K* из единиц. На практике увеличение размерности происходит по аналогии с размерами пулинга в свёрточных дискриминативных сетях и почти всегда равно *K*=2.

**Изображение выглядит как диаграмма, снимок экрана, дизайн

Контент, сгенерированный ИИ, может содержать ошибки.**

Таким образом, генератор в случае DCGAN является последовательностью свёрток, слоёв батч нормализации, нелинейностей и слоёв upsampling, а дискриминатор – обычной классификационной нейросетью. При этом первым слоем в генераторе является линейный слой, который отображает вектор шума *z* в карту признаков с начальным разрешением (как правило, размера 4×4).

## **Условная генерация**

Допустим, что в нашем датасете есть изображения, относящиеся к разным классам, и мы хотели бы уметь генерировать изображение заданного класса. В этом случае речь идёт об условной генерации. В качестве условия может выступать не только метка класса, но и объект любой природы. Например, когда вы можете захотеть сгенерировать изображение по текстовому описанию.

Далее будем обозначать условие как *y*. Наша задача — построить генератор, который бы моделировал *p*(*x*∣*y*).

**Conditional GAN**

Самый основной метод условной генерации — конкатенация условия с вектором шума, который генератор принимает на вход. В дискриминатор также подается условие на вход.

Изображение выглядит как снимок экрана, текст, диаграмма, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Если мы генерируем векторные данные, то вектор на вход дискриминатора подаётся конкатенированным с *y*. При этом если если *y* — это метка класса, то стоит её закодировать с помощью one-hot encoding. Если же мы работаем с изображениями, то нам из вектора условия следует сделать изображение.

**Современные модели**

Теперь на примере наиболее успешных моделей мы расскажем об улучшениях, которые во многом отходят от оригинального подхода к обучению GAN-ов и при этом значительно улучшают практические результаты, а значит расширяют практическую применимость.

## **StyleGAN**

Самой известной генеративно-состязательной моделью, работающей с изображениями, по праву считается StyleGAN, который до сих пор активно развивается и имеет большое количество расширений (например, существуют разнообразные методы его обращения).

Архитектура StyleGAN. Суть данной техники заключается в том, чтобы не сразу генерировать изображение высокого разрешения, а постепенно. Давайте рассмотрим это подробнее.

**Изображение выглядит как Человеческое лицо, человек, снимок экрана, улыбка

Контент, сгенерированный ИИ, может содержать ошибки.**

Мы хотим получить генератор, который генерирует изображения размера 1024x1024. Обучить такой генератор очень сложно. Поэтому мы начинаем с разрешения 4x4. У генератора мы оставляем только первый блок слоёв, который позволяет из шума получить изображение размера 4x4. У дискриминатора мы оставим, наоборот, только последний, который принимает на вход изображение размером 4x4. Такой GAN мы обучаем на изображениях из датасета (предварительно уменьшив их в размере).

Спустя сколько-то итераций мы понимаем, что сеть уже умеет генерировать маленькие изображения. В этот момент мы добавляем к генератору один блок, чтобы на выходе у неё получалось изображение размера 8x8. Так же мы добавляем один блок в начало дискриминатора, чтобы он на вход принимал изображения размера 8x8. Теперь генератор и дискриминатор состоят из двух блоков, которые мы и обучаем.

Такой процесс мы повторяем несколько раз, пока в итоге не дойдём до нужного нам разрешения 1024x1024.

**Подача шума в нейросеть**

Ключевой частью StyleGAN является используемый в нём способ подачи шума *z* в нейросеть, и именно из-за него метод и получил своё название. Для того чтобы понять, что конкретно в нём особенного, давайте подробнее посмотрим на архитектуру сети (рисунок из предыдущего раздела, модель StyleGAN справа).

Изображение выглядит как текст, диаграмма, снимок экрана, План

Контент, сгенерированный ИИ, может содержать ошибки.

Во-первых, вместо того, чтобы подавать вектор шума *z* только в самом начале генератора, нейросеть обуславливают на него много раз на разных разрешениях признаков. Исторически, впервые похожим методом решалась задача переноса стиля одной картинки на другую, отсюда и название: a style-based generator.

В качестве метода обуславливания используются так называемые адаптивные слои. Это модификация обычных слоёв нейросетей, в которых часть параметров предсказывается другой нейросетью. Вообще говоря, адаптивным можно сделать любой вид нормализации, включая батч нормализацию, но наиболее известным примером такого слоя является *адаптивная инстанс нормализация* (*adaptive instance normalization*), и именно она использовалась в первой версии StyleGAN.

## **StyleGAN-ADA**

**Аугментация данных** — это процесс искусственного увеличения объёма тренировочных данных путём создания модифицированных версий существующих примеров.

Следующий шаг в развитии архитектуры StyleGAN — это StyleGAN-ADA. ADA расшифровывается как Adaptive Discriminator Augmentation. Авторы предложили механизм аугментации данных, который позволяет стабилизировать обучение и избежать переобучения дискриминатора.

Изображение выглядит как текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

На левом рисунке (b) изображено, куда добавляется аугментация (синие блоки). На правом рисунке (c) изображена степень аугментации в зависимости от контролирующего её параметра p.

Основная идея алгоритма контроля *p* в процессе обучения следующая. Изначально этот параметр равен нулю. Его значение изменяется на фиксированную величину каждые четыре мини-батча. Если наблюдается, что дискриминатор слишком переобучился, то параметр *p* увеличивается. И наоборот, при низкой степени переобучении дискриминатора значение *p* уменьшается.

Аугментация, как показали авторы, действительно помогает стабилизировать обучение при маленьком количестве данных. Однако, большой набор реальных данных всегда будет выигрывать у аугментации.

# **Применения генеративных состязательных нейросетей**

До этого мы рассмотрели применение нейросетей в задаче генерации изображений. В этом разделе мы рассмотрим, какие ещё задачи можно решать.

Отметим, что задачи, которые мы рассмотрим ниже, можно решать и другими способами без ГАНов. Зачастую диффузионные модели (MidJourney, Stable Diffusion) показывают лучшие результаты в этих задачах. Тем не менее в данном же разделе мы рассмотрим именно методы на основе генеративных состязательных нейросетей.

## **Inpainting**

Представьте, что вы хотите удалить с фотографии людей на заднем плане. Встаёт вопрос, чем их заменить? Для этого существует задача инпеинтинга (inpainting). Она заключается в том, чтобы восстановить часть изображения, которая была выделена маской. Если выделить людей или объекты на фотографии маской, то нейросеть для инпеинтинга будет способна зарисовать эти участки чем-то подходящим для конкретной фотографии.



Обычно генератор модели GANs для инпеинтинга представляют собой image-to-image модели. То есть изображение подаётся как на вход, так и на выход. То, что происходит внутри генератора, зависит от архитектуры модели.

Чтобы обучить модель инпеинтинга, нужно подготовить данные в формате пар <изображение с маской, изображение без маски>. Достаточно случайным образом выделить участки для удаления, после чего обучать нейросеть их восстанавливать.

## **Outpainting**

Задачу inpainting можно так же превратить в задачу outpainting, то есть дорисовки изображения по краям. Для этого нужно в качестве маски подать пиксели, которые находятся за рамками изображения.

Изображение выглядит как вода, небо, снимок экрана, на открытом воздухе

Контент, сгенерированный ИИ, может содержать ошибки.

Задача outpainting может быть полезна, когда хочется расширить изображение, например, чтобы увеличить его разрешение.

## **Редактирование изображений**

До этого мы рассматривали, как можно редактировать латентное пространство обученной состязательной модели, чтобы это отражалось на сгенерированных изображениях. В 2023 году вышла работа Drag Your GAN, которая основана на этом принципе, и позволяет редактировать изображения перетаскиванием одной точки в другую.

Метод Drag Your GAN основан на модели StyleGAN2. Ему на вход подаётся набор изначальных точек и набор конечных точек. Внутри метода поочерёдно выполняются следующие два шага:

1. Обновление латентного пространства и обновления изображения с помощью оптимизации;
2. Обновление координат точек (трекинг точек).

Изначально метод работает только со сгенерированным изображениями. Однако, нет проблем в том, чтобы добавить кодировщик, который бы переводил реальные изображения в латентное пространство модели. В таком случае можно будет редактировать и реальные изображения.

