Санкт-Петербургский государственный университет

МИТРОФАНОВ Егор Владимирович

Выпускная квалификационная работа

Оптимизация генеративно-состязательных сетей

Направление 02.04.02

“Фундаментальная информатика и информационные технологии”

ООП “Цифровые технологии и системы”

Научный руководитель: кандидат технических наук, доцент кафедры КММС
Гришкин Валерий Михайлович

Рецензент: старший научный сотрудник, Общество с ограниченной ответственностью «Нокиа солюшнз энд нетворкс»
Епифанов Николай Анатольевич

Санкт-Петербург

2022

Содержание

Содержание 2

Введение 3

Обзор литературы 5

Постановка задачи 9

Основная часть 10

1. Архитектура генеративных сетей 10
   - 1.1 DC-GAN 10
   - 1.2 WGAN 11
   - 1.3 Conv2d 12
   - 1.4 ConvTranspose2d 12
   - 1.5 BatchNorm2d 13
   - 1.6 InstanceNorm2d 14
2. Разработка средств для тестирования методов оптимизации генеративных сетей 14
   - 2.1 Выбор средств 14
   - 2.2 Трансферное обучение 15
   - 2.3 Дистилляция знаний 16
   - 2.4 Оценка результатов 17
3. Экспериментальное исследование методов оптимизации генеративных сетей 19
   - 3.1 Трансферное обучение 19
   - 3.2 Дистилляция знаний 22
   - 3.3 Сравнение показателей 24

Заключение 26

Список использованных источников 27

Введение

В современном обществе компьютер позволяет облегчить жизнь человека в многих областях. Но последние исследования показывают, что с помощью нейросетей можно не только выполнять задачи классификации, сегментации и детекции, а создавать совершенно новые объекты из целевого признакового пространства. Однако, чтобы обучать модели выполняющие данную задачу требуется огромное количество ресурсов, которые не являются общедоступными.

Так, к примеру, одна из новейших генеративных моделей ruDALL-E XXL, которая умеет генерировать изображения по текстовому запросу, имеет 12 миллиардов параметров. Только чтобы хранить данное количество параметров в четырехбайтовом типе нужно более 44 гигабайт памяти, когда большинство современных GPU не имеют подобных мощностей.

a) b)

Рис. 1 Изображения сгенерированные ruDALL-E по запросам: a) заснеженный лес, b) лес в огне

Тема генеративных сетей только начала набирать популярность, поэтому на данный момент количество исследований и публикация по данной теме не так велико. Большинство исследований над генеративными сетями были проведены на датасете MNIST, содержащий рукописные цифры

размером 28x28 пикселей. Однако с увеличением разрешения и количества важных деталей объекта генерации задача усложняется в разы, так как сходимость генеративных сетей еще не до конца изучена.

В данной работе будет совершен переход на исследование более крупного разрешения в 128x128 пикселей и к более сложной задаче: генерация мордочек животных.

Целью данной работы является исследование двух методов оптимизации нейросетей в применении для задачи обучения генеративно-состязательных сетей (GAN) и возможности получения качественных моделей на обычных пользовательских персональных компьютерах.

Обзор литературы

В статье [1] рассказывается о новом на тот момент времени типе моделей: GAN (generative adversarial networks) представляют собой совокупность двух моделей - дискриминатор и генератор. В то время как генератор стремится генерировать объект наилучшего качества, дискриминатор пытается научиться отличать реальные объекты от сгенерированных. В последней части приводится теоретическое доказательство сходимости алгоритма, но в данной работе оно рассматриваться не будет. В самом простом случае в качестве дискриминатора выступает простой бинарный классификатор (тогда генеративная модель называется DC-GAN), который вычисляет вероятность того, что изображение принадлежит к классу сгенерированных. Более подробно архитектура генеративных сетей будет рассмотрена в основной части.

Работа [2] освещает проблемы самых первых генеративных сетей (таких как DCGAN) и вопросов их сходимости, и ее авторы предлагают совершенно новую модификацию на основе расстояния Вассерштейна (WGAN). В статье приведены как теоретические доказательства, так и результаты экспериментов, показывающих превосходство данного вида сетей над ее предшественниками. Подробнее об обучении WGAN будет рассказано в основной части.

В статье [3] авторы сравнивают производительность и сходимость различных генеративных сетей и предлагают улучшенный алгоритм для обучения WGAN, который и будет использован в экспериментальной части далее с оптимальными параметрами. Основная идея алгоритма заключается в добавлении штрафа для дискриминатора, чтобы он не мог сильно превзойти генератор и дальнейшее обучение стало невозможным. Таким образом на каждом шаге алгоритма нужно производить дополнительные вычисления,

однако это оправдано ускорением и стабилизацией сходимости, что было доказано в соответствующей работе.

Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters w_0, initial generator parameters θ_0.
1: **while** θ has not converged **do**
2:   **for** $t = 1, \ldots, n_{\text{critic}}$ **do**
3:     **for** $i = 1, \ldots, m$ **do**
4:       Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0,1]$.
5:       $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
6:       $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
7:       $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
8:     **end for**
9:     $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
10:   **end for**
11:   Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
12:   $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
13: **end while**

Рис. 2 Алгоритм обучения WGAN с gradient penalty

В публикации [4] речь идет об одном из основных методов оптимизации обучения моделей - трансферное обучение (transfer learning) в задаче оптимизации генеративных сетей. Суть данного метода состоит в заморозке некоторого количества слоев обучаемой модели и использовании весов из других предобученных моделей. Данная оптимизация позволяет экономить на времени обучения, а также использовать больше данных что соответственно хорошо сказывается на отображении моделями признакового пространства.

В статье [5] предлагается новый способ оптимизации моделей - дистилляция знаний (knowledge distillation). Данная оптимизация направлена не на улучшение процесса обучения, а на сокращение объема рабочей модели. Имея качественную, но очень тяжелую модель можно обучить новую модель более простой структуры максимально повторяющая ее поведение. Очевидно новая модель не сможет полностью обучиться имитировать

распределение оригинальной, однако будет намного компактнее, что так необходимо во многих современных исследованиях, к примеру создание компактных устройств распознавания речи для слабослышащих.

Автор публикации [6] предлагает использовать дистилляцию знаний для генеративных сетей. В качестве цели выступает уменьшение размеров генератора. Соответственно помимо исходного и сжатого генератора в данном методе также присутствует дискриминатор. Такая модель называется KDGAN. Генераторы корректируют свои веса между собой с помощью дистилляционной функции потерь и генеративной функции потерь между дискриминатором. Таким образом у автора получилось сохранить генерирующие свойства модели, уменьшив ее размер. На рис. 3 показано, что процесс обучения у KDGAN проходит намного эффективнее предыдущих фаворитов в данной области.

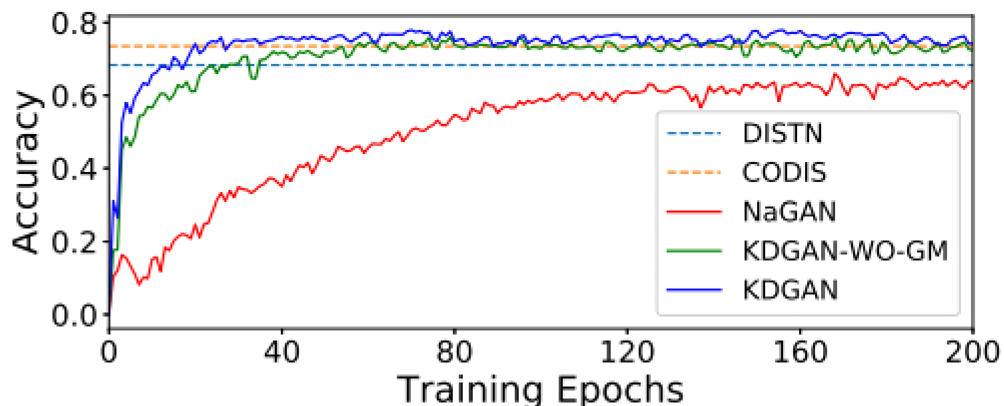

Рис. 3 Сравнение процесса обучения KDGAN на датасете MNIST

Еще одна из проблемных областей в обучении генеративных сетей - это оценка качества моделей. Действительно, в задаче генерации какого-либо распределения метрики использующиеся в стандартных задачах классификации, детекции и распознавания не будут иметь смысл. Один из наиболее очевидных и эффективных методов - это экспертная оценка. Однако данный метод имеет свои недостатки: большие затраты временных/людских ресурсов и предвзятость экспертов. В статье [7] рассказывается о метрике,

которая подходит для задачи оценки качества генеративных сетей. FID Score или Frechet Inception Distance - это метрика, предложенная в 2017 году, которая оценивает расстояние между признаками полученными с помощью некоторой вспомогательной нейросети (Inception network) для сгенерированной и реальной выборки. Обычно в качестве вспомогательной сети используется стандартная нейросеть Inceptionv3 предобученная на датасете ImageNet. FID Score между выборками с средним значением и стандартным отклонением (m, C) и (m_w, C_w) вычисляется как:

$$d^2((m, C), (m_w, C_w)) = \left\|m - m_w\right\|_2^2 + Tr(C + C_w - 2(CC_w)^{1/2}).$$

Все представленные публикации дают общее представление о необходимости исследования генеративных сетей в различных сферах, а также большое количество способов их модификации и оптимизации.

Постановка задачи

Провести сравнительный анализ следующих методов оптимизации нейронных сетей в применении к генеративно-состязательным сетям: трансферное обучение (transfer learning) и дистилляция знаний (knowledge distillation) по следующим признакам:

1. Экономия памяти при обучении модели
2. Размер рабочей модели
3. Качество работы модели

Качество работы модели оценивать по метрике FID score. Сделать вывод по преимуществам и недостаткам методов и для решения каких задач они лучше подходят.

Основная часть

1. Архитектура генеративных сетей

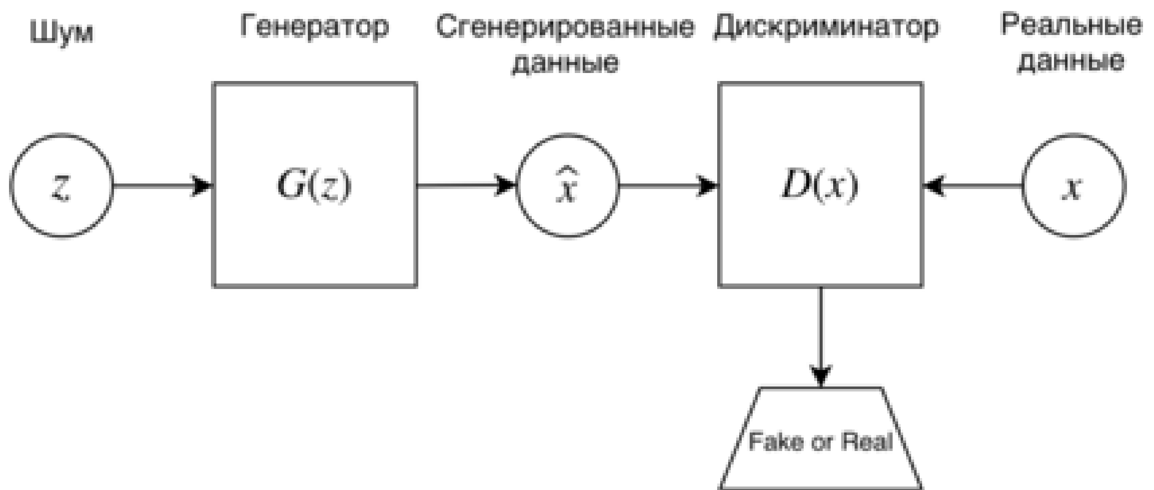

Рис. 1.1 Схема обучение GAN

1.1 DC-GAN

На рисунке 1.1 представлена схема обучения генеративных сетей. Чтобы научиться моделировать распределение реальных данных $X: [x \epsilon E^n]$ зададим дифференцируемую функцию $G(z; \theta_g)$, отображающую вектор $z \epsilon E^m$ взятый из заданного распределения p_z в вектор пространства E^n и имеющую параметры θ_g. Также зададим скалярную функцию $D(x; \theta_d)$ которая будет представлять вероятность что $x \epsilon X$. Таким образом мы можем произвести оптимизацию θ_g с помощью минимакс игры двух игроков G и D:

$$min(G)\, max(D)\, V(D, G) = E_{x \sim X}[log\, D(x)] + E_{z \sim p_z}[log(1 - D(G(z)))]$$

В случае с небольшими размерами пространства E^n в качестве функций G и D наиболее целесообразно использовать многослойные перцептроны. В случае же работы с многомерными пространствами, такими как изображения используются слои свертки и транспонированной свертки.

1.2 WGAN

Проблема DC-GAN заключается в том, что может произойти значительный перевес со стороны дискриминатора. В таком случае обучение может застывать в одной точке в самом начале и обучение генератора становится невозможным. Модификация WGAN (Wasserstein GAN) основана на использовании метрики Вассерштейна для того чтобы ввести пенальти по градиенту для дискриминатора и нивелировать данный перевес. В данном случае для дискриминатора не требуется выдавать вероятность: он может просто выдавать действительное число - оценку изображения, в таком случае дискриминатор называют также критиком. Функции потерь генератора (LossG) и дискриминатора (LossD) тогда будут выглядеть следующим образом:

$$LossG = E_{z\sim p_z}[D(G(z))],\ LossD = E_{z\sim p_z}[D(G(z))] - E_{x\sim X}[D(x)] + GP$$

$$GP = \lambda E_y[(\left\|\nabla_y D(y)\right\|_2 - 1)^2],\ y = \varepsilon x + (1 - \varepsilon)G(z),\ \varepsilon \in U[0, 1].$$

При реализации генератора и дискриминатора в данной работе были использованы вышеупомянутые слои сверток (Conv2d, ConvTranspose2d), а также слои BatchNorm2d и InstanceNorm2d, поэтому кратко рассмотрим принцип их работы.

1.3 Conv2d

Слои двумерной свертки - это хорошо известные и широко используемые в современных нейросетях модули. В качестве параметров обучения для данного слоя выступают ядра свертки (kernels) количество которых задается при инициализации. Каждое ядро свертки является матрицей заданного размера, при этом чаще всего применяются квадратные ядра (к примеру 3x3 или 5x5). Каждое значение выхода получается путем “наложения” ядра на вход и вычисления суммы произведений в соответствующих точках. На рис 1.2 проиллюстрирована работа сверточного слоя с ядром размера 3.

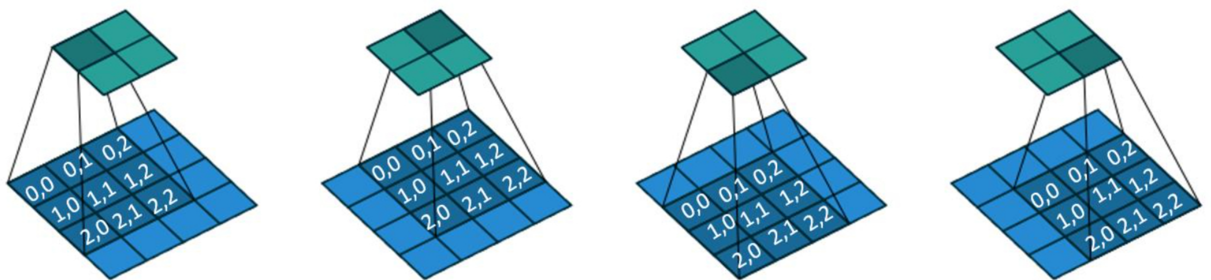

Рис 1.2 Пример работы сверточного слоя. Синим цветом обозначены входные данные, зеленым полученные на выходе

1.4 ConvTranspose2d

Слои транспонированной свертки или слои деконволюции начали получать широкое применение с развитием генеративных сетей и автоэнкодеров. Идея данного состоит в том, чтобы “обратить” эффект обыкновенного сверточного слоя уменьшения размерности изображения. Как и в случае с обыкновенным Conv2d используются ядра и происходит “наложение”, однако в данном случае в формировании угловых пикселей выхода будет участвовать всего один пиксель входа, то есть происходит смена ролей входных и выходных матриц. Принцип работы показан на рис 1.3

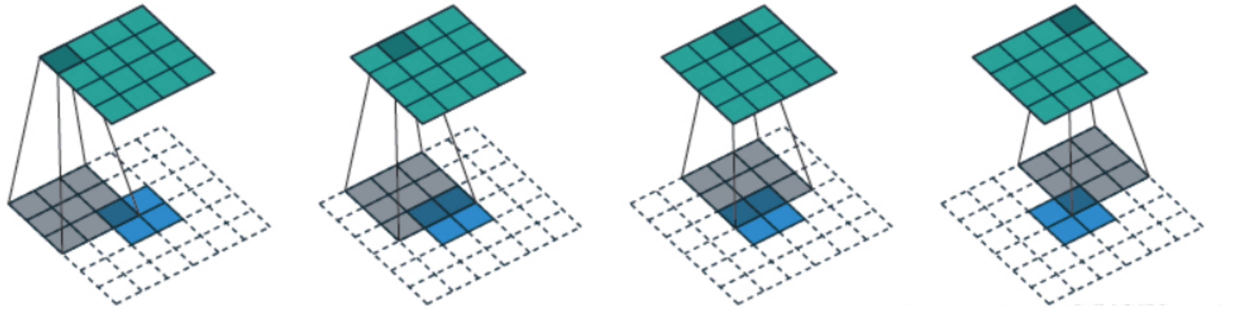

Рис 1.3 Пример работы транспонированного сверточного слоя. Синим цветом обозначены входные данные, зеленым полученные на выходе

1.5 BatchNorm2d

Слои пакетной нормализации стали незаменимым инструментом в построении современных глубоких нейросетей. Такой популярности данный метод достиг благодаря большому количеству полезных свойств. Данный метод позволяет не только решать проблему ковариационного сдвига, когда распределение данных в обучающей выборке не соответствует распределению данных в тестовой, но и ускорить обучение играя роль регуляризации, а также вносить некоторый шум в веса глубоких слоев, работая аналогично методу dropout. Работу данного слоя можно представить с помощью формулы, где среднее значение и стандартное отклонение вычисляются для всего батча данных:

$$y = \frac{x - E[x]}{\sqrt{Var(x) + \epsilon}} * \gamma + \beta$$

В данной формуле γ и β являются обучаемыми параметрами слоя, имеющие размерность равную количеству каналов входа.

1.6 InstanceNorm2d

Данный слой может быть описан точно такой же формулой как и пакетная нормализация, за исключением того, что стандартное отклонение и среднее значение вычисляются для каждого объекта в батче отдельно. Для дискриминатора в задаче обучения генеративных сетей с модификацией градиент пенальти пакетная нормализация будет не валидна, так как в ней теряется часть данных. Таким образом данная нормализация является наиболее целесообразной для данных моделей и будет будет использована в реализации.

2. Разработка средств для тестирования методов оптимизации генеративных сетей

2.1 Выбор средств

Для реализации экспериментальной системы был выбран язык Python 3.6, так как он является наиболее популярным для исследований в области машинного обучения и прост в установке и эксплуатации.

В качестве фреймворка машинного обучения был выбран pytorch, так как он является одним из самых популярных и имеет большое количество возможностей для контроля параметров и внесения изменений до самого глубокого уровня.

Для оценки работы системы на данном этапе были использованы датасеты с мордочками котов и собак взятые с ресурса kaggle.com

Датасет 1 состоит из двух частей. Первая из них [8]. Данный датасет содержит полные изображения котов и файлы с метками расположения ключевых точек. Для данной части была выполнена соответствующая обработка и выделены только мордочки. Вторая часть была взята из датасета

[9], где изображения уже нужного вида и разрешения 512x512 пикселей. Общий размер датасета 15000 изображений.

Датасет 2 был взят из того же источника что и вторая часть датасета 1 и содержит 6000 изображений мордочек собак с разрешением 512x512.

В качестве предобработки изображений они нормализуются к отрезку [-1;1] и приводятся к расширению 128x128.

Так как датасет ImageNet содержит в себе изображения животных, то для получения метрики FID Score будет целесообразно использовать стандартную вспомогательную сеть Inceptionv3.

Обучение моделей происходило на видеокарте ноутбука имеющего следующие характеристики:

1. видеокарта NVIDIA GeForce GTX 1660 Ti Max-Q 6gb
2. процессор AMD Ryzen 7 4800h with radeon graphics

2.2 Трансферное обучение

Для проведения эксперимента была выбрана пятислойная архитектура генератора и дискриминатора (см. в приложении). Далее была разработана система для обучения генеративной модели с модификацией gradient penalty и параметрами из работы [3].

В ходе эксперимента первым делом обучалась модель (далее **модель 1**) генерации кошачьих мордочек на датасете размером 15 тысяч изображений (далее **датасет 1**). Обучение происходило дольше всего с целью получения наиболее качественной модели.

Далее обучались две новые модели на датасете из 5 тысяч изображений собачьих мордочек (далее **датасет 2**). Первая модель была обучена с нуля, а для второй была взята предобученная на кошках модель и заморожены глубокие слои как в генераторе так и в дискриминаторе. Для дискриминатора было заморожено 4 слоя, для генератора 2 слоя.

Целью эксперимента было показать, что имея одну качественную модель можно получить новую на другом датасете со схожим признаковым пространством которая будет давать достаточно хорошее качество при меньших затратах ресурсов.

2.3 Дистилляция знаний

Дистилляция знаний является одним из новых но достаточно популярных методов оптимизации нейронных сетей, с помощью которого, используя имеющуюся хорошо обученную модель сложной архитектуры (учителя), происходит обучение новой модели более компактной архитектуры, при этом максимально стараясь сохранить производительность учителя. При стандартных задачах, таких как классификация и распознавание, где модель представлена одной нейросетью схема обучения выглядит как показано на рисунке 2.1. Функция потерь строится на различии выхода учителя и ученика. Наиболее простой и популярной мерой различия является среднеквадратическая ошибка (MSE Loss).

$$l(x, y) = L_{x,y} = \frac{1}{n}(l_1 + \dots + l_n),\ l_n = (x_n - y_n)^2$$

Для проведения эксперимента была использована модель 1 из части 3.1 в качестве учителя. В качестве обучаемой модели архитектура генератора из части 3.1 была упрощена убрав один из слоев и уменьшив количество параметров слоев транспонированной свертки. В архитектуре дискриминатора была добавлена сигмоидная активация. В качестве функции потерь будем использовать предлагаемую в публикации [6] функцию JointLoss сочетающую генеративную функцию потерь для DC-GAN и среднеквадратическую с параметром α, отвечающим за вклад каждой из ошибок в общую функцию потерь.

$$j(x, y) = J = E_{z\sim p_z}[log(1 - D(G(z))] + \alpha * E_{x\sim X}[log(D(x))] + (1 - \alpha) L_{x,z}$$

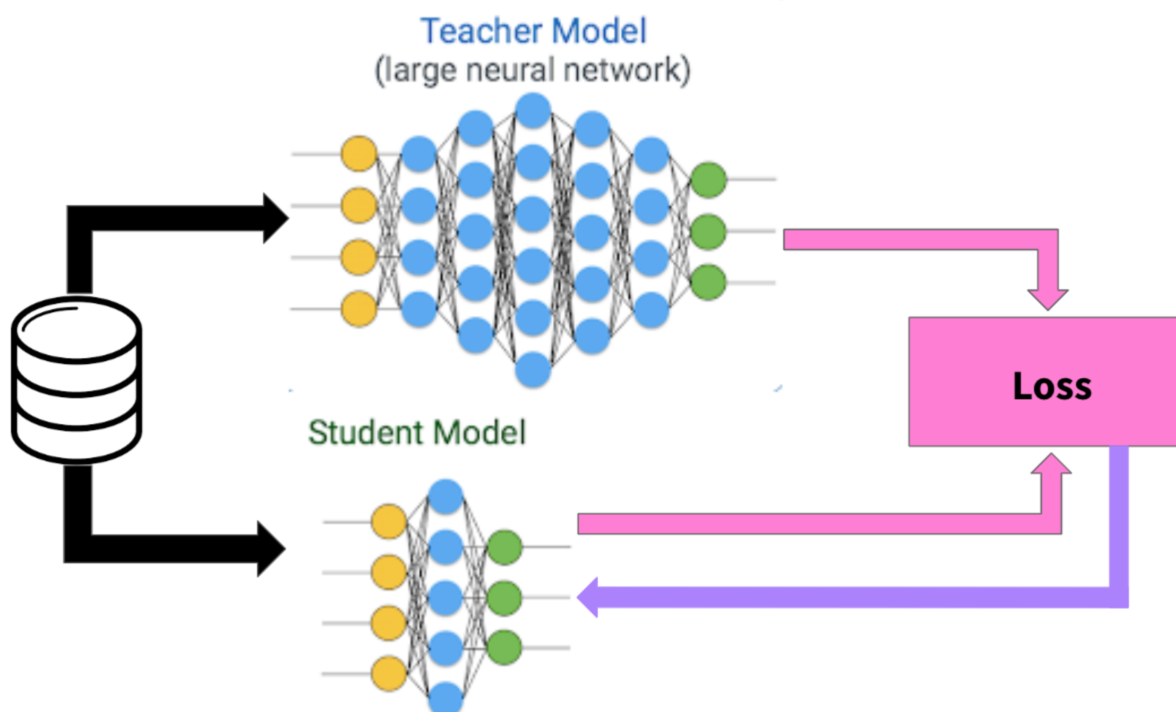

Рис 2.1 Схема обучения с использованием дистилляции знаний

Так как в публикации не было представлено оптимального параметра α, были проведены эксперименты для нескольких значений.

2.4 Оценка результатов

Для оценки результатов был реализован соответствующий скрипт **createsamples.py**. Данный скрипт берет архитектуру генератора из **arch.py** и имя сохраненной модели и создает тестовую выборку заданного размера подавая на вход модели вектора случайно взятые из нормального распределения. Для всех экспериментов размер выборки был равен 500. Репозиторий с исходным кодом скриптов и обучения: https://github.com/MitrofanovEV/GAN_Optimization.

После обучения каждой из моделей написанным скриптом генерировалась выборка из 500 изображений. Из датасета 1 была также сделана выборка соответствующего размера.

Метрика FID рассчитывалась с помощью библиотеки pytorch-fid (https://github.com/mseitzer/pytorch-fid). На вход скрипту для расчета метрики подаются две папки содержащие 500 изображений исходного распределения (датасет 1) и 500 сгенерированных моделью изображений.

Для того, чтобы наглядно показать работу моделей заранее были зафиксированы шесть 100 мерных векторов из нормального распределения. При обучении каждой модели в процессе обучения для данных шести векторов получали сгенерированные изображения. Таким образом можно наблюдать постепенное улучшение качества за счет обновлений градиента.

Алгоритм WGAN требует более часто обновлять градиент дискриминатора, поэтому одна эпоха при обучении WGAN в реализации состоит из 200 обновлений дискриминатора и 25 обновлений генератора, при этом при алгоритме DC-GAN одна эпоха состоит из 100 обновлений градиента дискриминатора и 100 генератора.

3. Экспериментальное исследование методов оптимизации генеративных сетей

3.1 Трансферное обучение

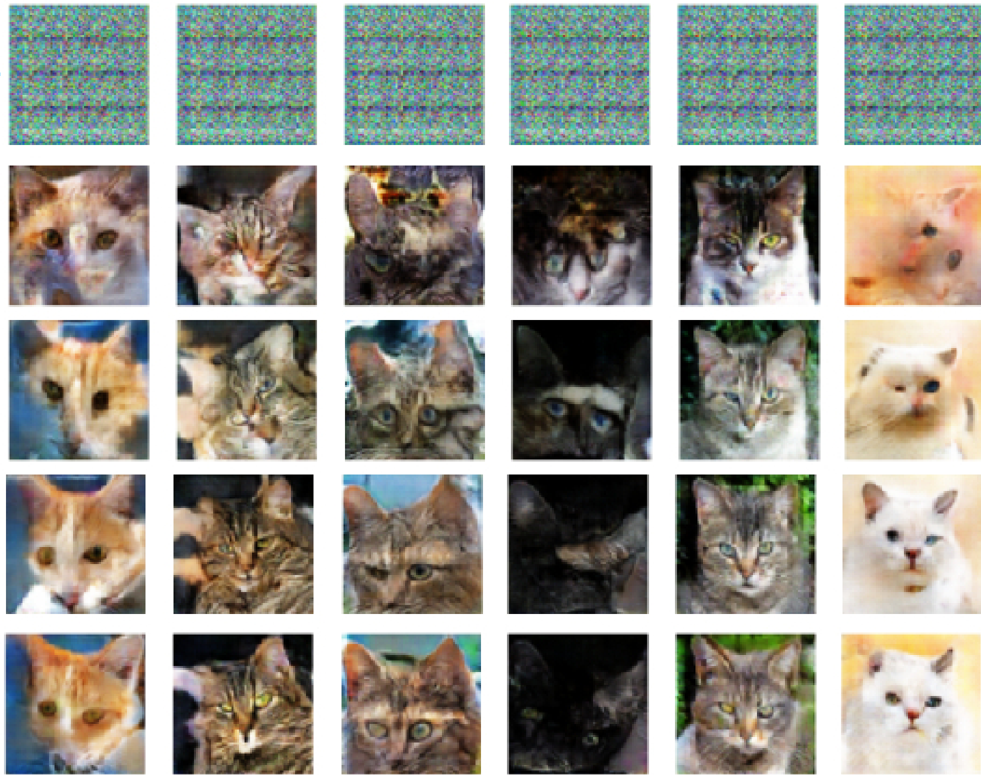

Рис 3.1. иллюстрация обучения генератора для входного фиксированного вектора для модели 1 с интервалом в 1000 эпох

Рис 3.2. Сравнение работы a) модели дообученной на основе модели 1 на датасете 2 и b) модели обученной с нуля на датасете 2 с интервалом в 200 эпох

На рис 3.1 можно наблюдать постепенное улучшение качества генерации в процессе обучения модели 1 для заранее выбранного вектора. Сначала веса модели случайные и получается лишь зашумленное изображение. Далее постепенно проявляется очертания объектов и они становятся все более четкими. Для модели 1 решено было остановиться на 4000 эпохах, так как и визуальное качество и значение метрики FID перестали заметно расти по сравнениею с 3000.

Как можно заметить из рис. 3.2, способ с использованием дообучения модели, обученной на другом датасете, показывает себя более стабильно на данном количестве эпох, однако не достигает идеальной сходимости.

Еще одним замечанием, которое можно выделить является формирование новых объектов при дообучении. Так как глубокие слои сети были заморожены, новые объекты генерируемые из соответствующих входных векторов будут обладать схожими глобальными признаками с объектами оригинальной модели (цвет шерсти, направление взгляда, поворот

головы), в то время как при обучении с нуля эти признаки определяются для входных векторов уже во время обучения.

Основным преимуществом дообучения является значительная экономия вычислительных ресурсов, что открывает возможность использовать более сложные модели на слабых устройствах.

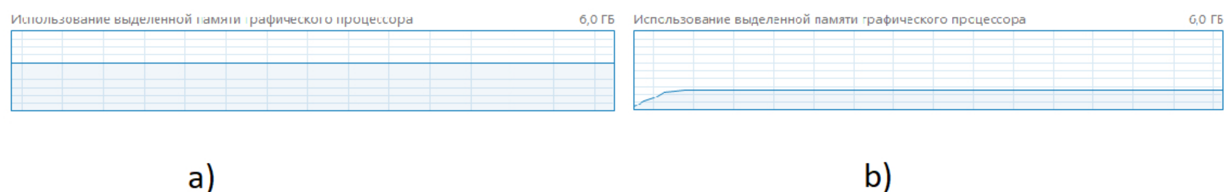

Рис 3.3 сравнение занимаемой видеопамяти при а) обучении с нуля, b) дообучении модели

Как видно из Рис. 3.3 дообучение модели может давать выигрыш более чем в 2 раза при заморозке большого когличества слоев, но при этом выдавать достаточно качественные изображения.

Рис 3.4 Пример работы генератора на случайных векторах

	FID score	1 эпоха обучения, сек	видеопамять при обучении, гб
модель 1	63.95	59.5	4.03
модель 2 (полная)	114.58	59.8	4.03
модель 2 (transfer learning)	127.517	46.2	1.45

Таблица 3.1 Результаты применения трансферного обучения

В таблице 3.1 приведены результаты проведенного эксперимента. Так как датасет 2 намного меньше датасета 1, качество метрики FID в сравнении с трансферным обучением значительно снизилось, поэтому в финальной части будет оцениваться дельта метрики в процентах.

3.2 Дистилляция знаний

Были обучены несколько моделей для различных показателей значений α. Результаты представлены в таблице 3.2.

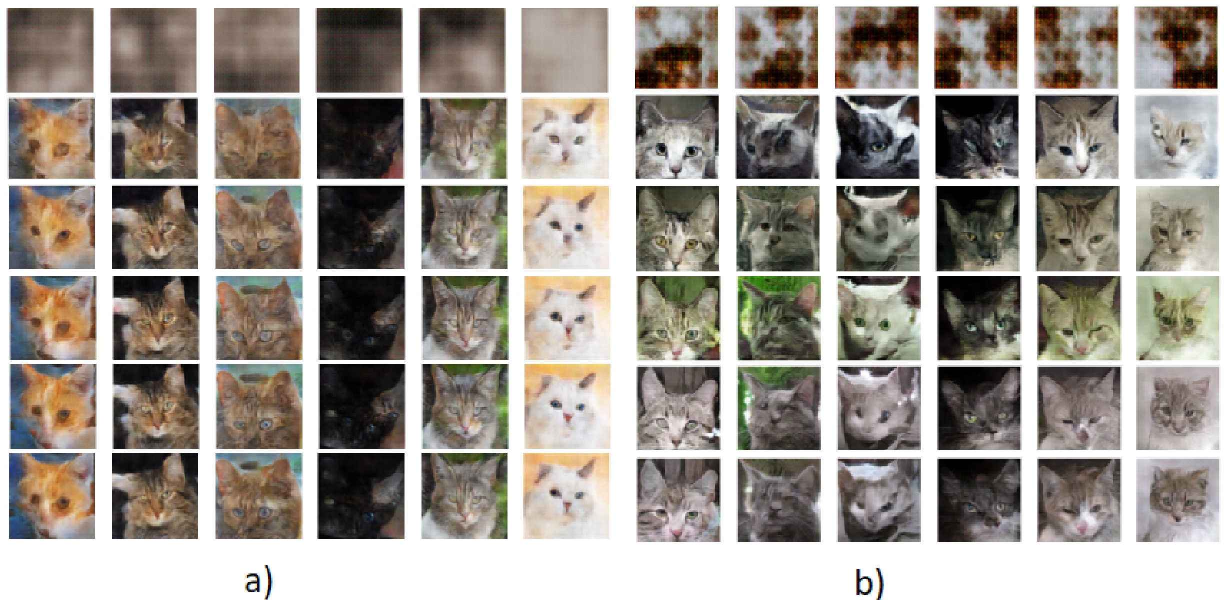

Рис 3.5 Процесс обучения генеративной модели с учителем a) α = 0.001, b) α = 0.3

Рис 3.6 Сравнение результатов работы моделей с различными параметрами a) α = 0.3, b) α = 0.05, c) α = 0.1, d) α = 0.001, e) учитель (модель 1)

	FID score с датасетом 1	FID score с учителем	одна эпоха обучения, сек	видеопамять при обучении, гб	α
модель KD 1	65.33	32.07	34	1.89	0.3
модель KD 2	68.69	29.8	34	1.89	0.05
модель KD 3	68.57	31.06	34	1.89	0.01
модель KD 4	79.38	30.94	34	1.89	0.001

Таблица 3.2 Результаты применения дистилляции знаний

Рис 3.5 показывает процесс обучения генеративной модели с помощью дистилляции знаний для двух различных значений параметра α для заранее выбранных входных векторов. Через каждые 200 эпох обучения фиксируются изображения выдаваемые генератором. Можно заметить, что при значении параметра 0.001 модель полностью стремится повторить поведение учителя, когда при значении 0.3 наблюдается изменение свойств модели. FID score при этом практически не отличается при сравнении с учителем, однако с датасетом 1 реальных изображений котов лучше всего себя показала модель KD 1. Стоит отметить, что хоть качество генерируемых изображений стало

лучше, то их цветовая гамма сместилась к более серым тонам и часть информации обучающей выборки была утеряна. Наиболее оптимальным вариантом судя по экспериментальным данным является модель KD 2 с параметром $\alpha = 0.05$: она не только сохраняет точную генерацию изображений, но и имеет вторую по качеству метрику FID к датасету 1.

На Рис. 3.6 показывается результат работы каждой из моделей после обучения на заранее выбранных векторах. Видно, что при больших значениях параметра α где будет преобладать стандартная функция потерь, модель во многих местах исправляет дефекты учителя, однако теряет в разнообразии генерируемых пород кошек. То есть наиболее эффективным будет являться такое значения параметра при котором среднеквадратичная ошибка будет достаточно сильно влиять на общую функцию потерь, чтобы сохранить распределение генерируемых данных, но недостаточно сильно, чтобы сохранять недоработки и дефекты оригинальной модели.

С точки зрения времени на одну эпоху обучения (100 обновлений градиента дискриминатора и генератора) данный метод также показывает преимущество над обычным обучением.

3.3 Сравнение показателей

Далее представлена сводная таблица по результатам экспериментов. Гипотеза о применимости данных методов оптимизации для генеративно-состязательных сетей подтвердилась. Данные методы не мешают сходимости моделей, а также показывают хорошие результаты и с точки зрения оптимизации времени обучения и в экономии памяти при обучении.

Одним из минусов обоих оптимизаций является несомненно потеря качества за счет уменьшения количества параметров или заморозки слоев, однако данные потери оправданы, ведь оптимизации дают огромный выигрыш в занимаемой видеопамяти, что и является основным дефицитом для пользователей в наше время.

В качестве оценки качества будем использовалась дельта метрики качества FID в процентах по отношению к обыкновенному обучению, так как размеры датасетов были различные, а это сильно влияет на качество обучения.

	Занимаемая память при обучении модели, гб	Занимаемая память рабочей модели, мб	Δ метрики качества FID, %
трансферное обучение	1.45	186	-11.2
дистилляция знаний	1.89	38	-7.4
обыкновенное обучение	4.03	186	0

Таблица 3.3 Сравнительный анализ методов оптимизации

Исходя из проведенных экспериментов видно, что трансферное обучение требует меньшее количество вычислительных ресурсов при достаточном коэффициенте сохранения качества, однако не уменьшает затраты на хранения самой модели, что может быть важно в некоторых задачах. Еще одно из преимуществ трансферного обучения, что оно не требует наличия качественной модели генерируемого распределения, можно взять веса любой модели где низкоуровневые признаки схожи.

Заключение

Трансферное обучение может применяться, когда имеется модель с похожим пространством признаков и позволяет добиться колоссальной экономии памяти за счет заморозки слоев. Такой метод практически всегда более эффективен чем обучение с нуля и очень часто применяется для таких задач как классификация и распознавание. Было показано, что данный метод дает хороший результат и для генеративных сетей и может применяться в таких задачах как генерация дипфейков или картин художников.

Для дистилляции знаний тоже были получены положительные результаты. Данная оптимизация позволяет значительно сократить размер самой генеративной модели, что может позволить портировать большие модели на мобильные и компактные устройства. Это может быть полезно, например при необходимости перенести одну большую модель на огромное количество смартфонов, при этом для каждого сохранив одинаковое распределение генерируемых данных.

В данной работе были рассмотрены два наиболее распространенных и востребованных метода оптимизации для современных нейронных сетей и их применение в задаче обучения генеративно состязательных сетей. Были проведены эксперименты и выполнен сравнительный анализ на основе трех параметров, а также предложены задачи для которых данные методы будут полезны.

Список использованных источников

[1] – Generative Adversarial Nets

URL - https://arxiv.org/pdf/1406.2661.pdf

[2] - Wasserstein GAN

URL - https://arxiv.org/pdf/1701.07875.pdf

[3] - Improved Training of Wasserstein GANs

URL - https://arxiv.org/pdf/1704.00028.pdf

[4] - Transfer Learning for GANs

URL - https://arxiv.org/pdf/1906.11613.pdf

[5] - Distilling the Knowledge in a Neural Network

URL - https://arxiv.org/pdf/1503.02531.pdf

[6] - KDGAN: Knowledge Distillation with Generative Adversarial Networks

URL - https://proceedings.neurips.cc/paper/2018/file/019d385eb67632a7e958e23f24bd07d7-Paper.pdf

[7] - GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium

URL - https://arxiv.org/pdf/1706.08500.pdf

[8] - Cat Dataset - Over 9,000 images of cats with annotated facial features

URL - https://www.kaggle.com/datasets/crawford/cat-dataset

[9] - Animal Faces - 16,130 images belonging to 3 classes.

URL - https://www.kaggle.com/datasets/andrewmvd/animal-faces

Приложение

```
Discriminator(
  (layer1): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): InstanceNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (layer2): Sequential(
    (0): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): InstanceNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (layer3): Sequential(
    (0): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (layer4): Sequential(
    (0): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): InstanceNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (layer5): Sequential(
    (0): Conv2d(512, 1024, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): InstanceNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (output): Sequential(
    (0): Conv2d(1024, 1, kernel_size=(4, 4), stride=(1, 1))
  )
)
```

модель 1-2 дискриминатор

```
Generator(
  (layer1): Sequential(
    (0): ConvTranspose2d(100, 2048, kernel_size=(4, 4), stride=(1, 1))
    (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (layer2): Sequential(
    (0): ConvTranspose2d(2048, 1024, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (layer3): Sequential(
    (0): ConvTranspose2d(1024, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (layer4): Sequential(
    (0): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (layer5): Sequential(
    (0): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (final): Sequential(
    (0): ConvTranspose2d(128, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  )
  (output): Tanh()
)
```

модель 1-2 генератор

```
Discriminator(
  (layer1): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): InstanceNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (layer2): Sequential(
    (0): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): InstanceNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (layer3): Sequential(
    (0): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (layer4): Sequential(
    (0): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): InstanceNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (layer5): Sequential(
    (0): Conv2d(512, 1024, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): InstanceNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (output): Sequential(
    (0): Conv2d(1024, 1, kernel_size=(4, 4), stride=(1, 1))
    (1): Sigmoid()
  )
)
```

KD модели дискриминатор

```
GeneratorKD(
  (layer1): Sequential(
    (0): ConvTranspose2d(100, 512, kernel_size=(4, 4), stride=(1, 1))
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (layer2): Sequential(
    (0): ConvTranspose2d(512, 256, kernel_size=(8, 8), stride=(4, 4), padding=(2, 2))
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (layer3): Sequential(
    (0): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (layer4): Sequential(
    (0): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (final): Sequential(
    (0): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  )
  (output): Tanh()
)
```

KD модели генератор