

МИНОБРНАУКИ РОССИИ  
Федеральное государственное автономное образовательное учреждение высшего  
образования  
«Национальный исследовательский университет  
«Московский институт электронной техники»

Факультет Микроприборов и технической кибернетики  
Кафедра Высшей математики №1

**Ильютченко Павел Сергеевич**

Магистерская диссертация  
по направлению 01.04.04 «Прикладная математика»

**«Повышение качества изображений с помощью  
нейронных сетей»**

Студент

\_\_\_\_\_

Ильютченко П.С.

Научный руководитель,

\_\_\_\_\_

к.ф.-м.н.

Козлитин И.А.

Москва 2019

# Оглавление

1. Введение .....	5
1.1 Определения .....	12
Свёртка .....	13
Скрытые блоки .....	20
Пулинг .....	23
1.2 Современные методы супер-разрешения.....	26
Интерполяция методом ближайшего соседа.....	26
Бикубическая интерполяция .....	26
Методы на основе разреженного кодирования.....	27
2. Супер-разрешения с помощью свёрточных нейронных сетей .....	30
2.1 Введение .....	30
2.2 Описание метода .....	33
Извлечение и представление патчей .....	34
Нелинейное отображение.....	35
Реконструкция .....	36
2.3 Обучение.....	37
2.4 Эксперименты.....	38
2.5 Заключение .....	42
3. Генеративно-состязательные сети.....	43
3.1 Введение .....	43
3.2 Главная идея .....	44
3.3 Алгоритм обучения .....	45
3.4 Эксперименты.....	46

3.5 Преимущества и недостатки .....	48
4. Практическое использование генеративно-состязательных сетей .....	49
4.1 Введение .....	49
4.2 Модель .....	50
4.3 Задача нахождения параметров нормального распределения .....	53
4.4 Задача приближения смеси нормальных распределений .....	55
4.5 Задача приближения распределения изображений .....	58
4.6 Итоги .....	62
5. Супер-разрешения с помощью свёрточных генеративно-состязательных сетей	62
5.1 Введение .....	62
5.2 Метод .....	64
Архитектура состязательной сети .....	65
Функция потери восприятия .....	67
5.3. Эксперименты.....	70
Данные и измерения сходства .....	70
Детали обучения и параметры .....	70
Изучение потери содержания .....	71
Результаты финальных сетей .....	73
5.4 Выводы.....	75
6. Усовершенствованные генеративно-состязательные сети супер-разрешения ...	76
6.1 Введение .....	76
6.2 Предложенный метод.....	78
Архитектура сети.....	79
Релятивистский дискриминатор .....	81
Потеря восприятия .....	82

Сетевая интерполяция .....	83
6.3 Эксперименты.....	84
Детали обучения .....	84
Данные .....	85
Качественные результаты .....	86
Исследование аблляции .....	88
Сетевая интерполяция .....	92
6.4 Выводы.....	93
7. Использование представления на основе формы в задаче супер-разрешения ...	94
7.1 Объяснение проблемы .....	94
7.2 Выводы авторов.....	95
7.3 Использование в супер-разрешении.....	96
8. Заключение .....	96
9. Источники.....	97

# 1. Введение

Впервые, программируя вычислительные машины, Ада Лавлейс (1842 г.) задумалась, смогут ли они стать разумными, и это было за сотню лет до создания компьютера. В наши дни **искусственный интеллект (ИИ)** – бурно развивающаяся дисциплина, имеющая множество прикладных задач. Все люди мечтают иметь интеллектуальные программы, которые будут автоматизировать их рутинные проблемы, понимать речь и изображения, ставить медицинские диагнозы и заниматься научными исследованиями.

Когда появились первые идеи об искусственном интеллекте, были быстро решены некоторые задачи, трудные для человека, но простые для компьютеров – описываемые с помощью списка формальных математических правил. А большой проблемой для ИИ стали задачи, которые легко решаются человеком, но с трудом поддаются формализации, например, распознавание устной речи или лиц на изображении.

Цель таких задач заключается в том, чтобы компьютер мог учиться на опыте и понимать мир в понятиях, которые определены через более простые понятия. Из-за приобретения знаний опытным путём этот подход избегает этап формального описания человеком всех необходимых знаний для компьютера. А иерархическая структура дает возможность учиться сложным понятиям через более простые. Всю такую структуру можно изобразить как граф, который будет очень глубоким – содержащим много уровней. Такой подход в ИИ называется **глубоким обучением**.

Ранее успехи компьютера были достигнуты в искусственной среде, где от компьютера не требовались обширные знания о мире. Например, созданная IBM, шахматная программа Deep Blue, которая в 1997 году обыграла чемпиона мира Гарри Каспарова. Шахматы – это ограниченный, достаточно простой мир, состоящий всего из 64 клеток и 32 фигур, которые ходят определенным образом. Разработка успешной стратегии игры в шахматы – это огромное достижение, но эта задача не трудна для

компьютера, если есть формальный список правил, которые заранее созданы программистом.

Компьютеры уже давно способны обыграть гроссмейстеров, но лишь в последние годы их способности сопоставимы с человеческими в части распознавания объектов или речи. В обычной жизни человеку нужен гигантский объем знаний о мире. Все эти знания субъективные и представлены в интуитивном виде, поэтому выразить их достаточно затруднительно для программиста. Но чтобы решать «разумные» задачи компьютерам необходимы такие знания. Основополагающей задачей для искусственного интеллекта является то, как заложить эти неформальные знания в компьютер.

Первые проекты в области ИИ пытались представить знания о мире с помощью формальных языков. Компьютер, используя логические правила вывода, может автоматически рассуждать о предложениях. В основе таких подходов лежит **база знаний**. Ни один из этих проектов не привел к существенному успеху.

Проблемы таких систем наводят на мысль, что система с искусственным интеллектом должна самостоятельно извлекать знания, отыскивая закономерности в исходных данных. Это умение называется **машичным обучением**. С появлением машинного обучения перед компьютерами открылась возможность решать задачи, требующие знаний о реальном мире, и принимать поддельные субъективные решения.

Качество таких алгоритмов зависит от представления исходных данных. Эта зависимость от представления является общим явлением, проявляющимся как в информатике, так и в повседневной жизни. Если говорить об информатике, то такие операции, как поиск записи в базе данных, будут производиться многократно быстрее, если база структурирована и индексирована. Неудивительно, что выбор представления оказывает огромное влияние на качество и производительность алгоритмов машинного обучения.

Многие задачи можно решить, если правильно подобрать признаки, а затем обучить алгоритм машинного обучения. Но во многих задачах сложно понять, какие признаки нужно выделять. Одно из решений этой проблемы – использовать машинное

обучение не только для того, чтобы найти отображение представления на результат, но, и чтобы определить само представление. Такой подход называется **обучением представлений**. На представлениях, полученных в ходе обучения, часто удается добиться гораздо более высокого качества, чем на представлениях, созданных вручную. К тому же это позволяет системам ИИ быстро адаптироваться к новым задачам при минимальном вмешательстве человека. Для простой задачи алгоритм обучения представлений может найти хороший набор признаков за несколько минут, для сложных – за время от нескольких часов до нескольких месяцев. Проектирование признаков вручную для сложной задачи требует много времени и труда, на это могут уйти десятилетия работы всего сообщества исследователей.

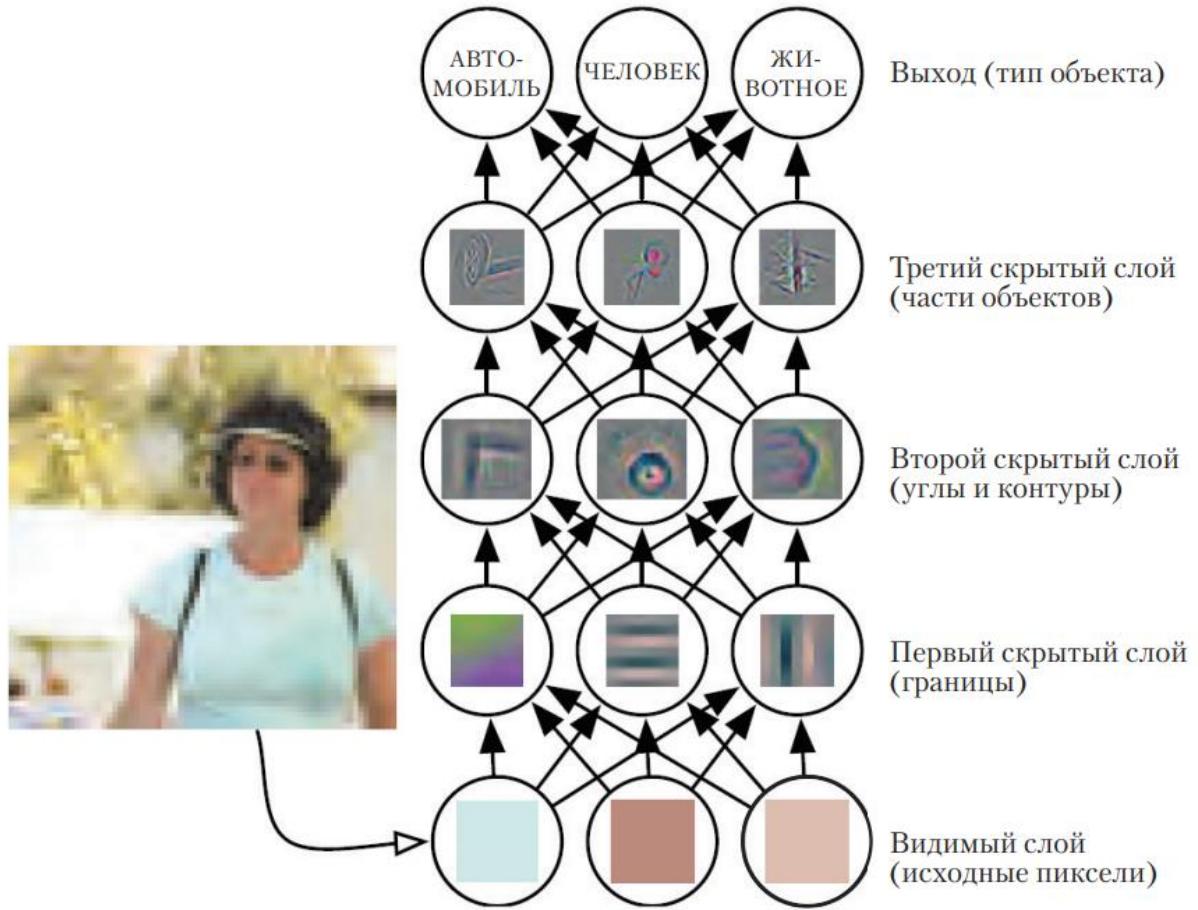
Квинтэссенцией алгоритма обучения представлений является **автокодировщик**. Это комбинация функции **кодирования**, которая преобразует входные данные в другое представление, и функции **декодирования**, которая преобразует новое представление в исходный формат. Обучение автокодировщиков устроено так, чтобы при кодировании и обратном декодировании сохранялось максимально много информации, но чтобы при этом новое представление обладало различными полезными свойствами. Различные автокодировщики ориентированы на получение различных свойств.

При проектировании признаков или алгоритмов обучения признаков целью обычно является выделение **факторов вариативности**, которые объясняют наблюдаемые данные. В этом контексте слово «фактор» означает просто источник влияния, а не «сомножитель». Зачастую факторы – это величины, не наблюдаемые непосредственно. Это могут быть ненаблюдаемые объекты или силы в физическом мире, оказывающие влияние на наблюдаемые величины. Это могут быть также умозрительные конструкции, дающие полезные упрощающие объяснения или логически выведенные причины наблюдаемых данных. Их можно представлять себе, как концепции или абстракции, помогающие извлечь смысл из данных, характеризуемых высокой вариативностью.

Источник трудностей в целом ряде практических приложений искусственного интеллекта – тот факт, что многие факторы вариативности оказывают влияние абсолютно на все данные, доступные нашему наблюдению. Отдельные пиксели изображения красного автомобиля ночью могут быть очень близки к черному цвету. Форма силуэта автомобиля зависит от угла зрения. В большинстве приложений требуется разделить факторы вариативности и отбросить те, что нам не интересны.

Разумеется, может оказаться очень трудно выделить такие высокоуровневые абстрактные признаки из исходных данных. Многие факторы вариативности, к примеру акцент говорящего, можно идентифицировать, только если наличествует очень глубокое, приближающееся к человеческому пониманию природы данных. Но раз получить представление почти так же трудно, как решить исходную задачу, то, на первый взгляд, обучение представлений ничем не поможет.

**Глубокое обучение** решает эту центральную проблему обучения представлений, вводя представления, выражаемые в терминах других, более простых представлений. Глубокое обучение позволяет компьютеру строить сложные концепции из более простых. На Рисунке 1 показано, как в системе глубокого обучения можно представить концепцию изображения человека в виде комбинации более простых концепций – углов и контуров, – которые, в свою очередь, определены в терминах границ.



*Рисунок 1. Иллюстрация модели глубокого обучения.*

Типичным примером модели глубокого обучения является глубокая сеть прямого распространения, или **многослойный перцептрон** (МСП). Многослойный перцептрон – это просто математическая функция, отображающая множество входных значений на множество выходных. Эта функция является композицией нескольких более простых функций. Каждое применение одной математической функции можно рассматривать как новое представление входных данных.

Идея нахождения подходящего представления данных путем обучения – это лишь один взгляд на глубокое обучение. Другой взгляд состоит в том, что глубина позволяет обучать многошаговую компьютерную программу. Каждый слой представления можно мыслить себе, как состояние памяти компьютера после параллельного выполнения

очередного набора инструкций. Чем больше глубина сети, тем больше инструкций она может выполнить последовательно. Последовательное выполнение инструкций расширяет возможности, поскольку более поздние инструкции могут обращаться к результатам выполнения предыдущих.

При таком взгляде на глубокое обучение не всякая информация, используемая для активации слоев, обязательно кодирует факторы вариативности, объясняющие входные данные. В представлении хранится также вспомогательная информация о состоянии, помогающая выполнить программу, способную извлекать смысл из данных. Эту информацию можно уподобить счетчику или указателю в традиционной компьютерной программе. Она не имеет никакого отношения к содержанию входных данных, но помогает модели в организации их обработки.

Итак, глубокое обучение – один из подходов к ИИ. Конкретно, это вид машинного обучения – методики, которая позволяет компьютерной системе совершенствоваться по мере накопления опыта и данных. На текущий момент машинное обучение – единственный жизнеспособный подход к построению систем ИИ, которые могут функционировать в сложных окружающих условиях. Глубокое обучение – это частный случай машинного обучения, позволяющий достичь большей эффективности и гибкости за счет представления мира в виде иерархии вложенных концепций, в которой каждая концепция определяется в терминах более простых концепций, а более абстрактные представления вычисляются в терминах менее абстрактных.

В настоящее время нейробиология рассматривается как важный источник идей для исследований в области глубокого обучения, но уже не занимает доминирующих позиций. Главная причина снижения роли нейробиологии в исследованиях по глубокому обучению – тот факт, что у нас попросту не хватает информации о мозге, чтобы использовать ее в качестве образца и руководства к действию. Чтобы по-настоящему понять алгоритмы работы мозга, нужно было бы одновременно наблюдать за активностью тысяч (как минимум) взаимосвязанных нейронов. Не имея такой

возможности, мы далеки от понимания даже самых простых и хорошо изученных областей мозга.

Нейробиология дала надежду на то, что один алгоритм глубокого обучения сможет решить много разных задач. Нейробиологи выяснили, что хорьки могут «видеть» с помощью области мозга, отвечающей за обработку слуховой информации, если перенаправить нервы из глаз в слуховую кору (Von Melchner et al., 2000). Это наводит на мысль, что значительная часть мозга млекопитающих, возможно, использует единый алгоритм для решения большинства задач. До появления этой гипотезы исследования по машинному обучению были более фрагментированы: обработкой естественных языков, компьютерным зрением, планированием движения и распознаванием речи занимались разные группы ученых. Эти сообщества и по сей день разделены, но ученые, работающие в области глубокого обучения, зачастую занимаются многими или даже всеми этими предметами.

Мы можем почертнуть из нейробиологии кое-какие соображения. Основная идея, навеянная осмыслением работы мозга, – наличие большого числа вычислительных блоков, которые обретают разум только в результате взаимодействий. Неокогнитрон (Fukushima, 1980) предложил архитектуру модели, эффективную для обработки изображений. Идея сложилась под влиянием структуры зрительной системы млекопитающих и впоследствии легла в основу современной сверточной сети (LeCun et al., 1998b). Большинство современных нейронных сетей основано на модели нейрона, которая называется **блоком линейной ректификации**. Оригинальная модель когнитрона (Fukushima, 1975) была более сложной, основанной на наших знаниях о работе мозга. В современной упрощенной модели объединены различные точки зрения; в работах Nair and Hinton (2010) и Glorot et al. (2011a) отмечено влияние нейробиологии, а в работе Jarrett et al. – скорее, инженерных дисциплин. Каким бы важным источником идей ни была нейробиология, считать, что от нее нельзя отклониться ни на шаг, вовсе необязательно. Мы знаем, что настоящие нейроны вычисляют совсем не те функции, что блоки линейной ректификации, но большее приближение к реальности пока не привело к повышению

качества машинного обучения. Кроме того, хотя нейробиология легла в основу нескольких успешных архитектур нейронных сетей, мы до сих пор знаем о биологическом обучении недостаточно, чтобы полнее использовать нейробиологию для построения алгоритмов обучения этих архитектур.

Современное глубокое обучение черпает идеи из разных дисциплин, особенно из таких отраслей прикладной математики, как линейная алгебра, теория вероятностей, теория информации и численная оптимизация. Некоторые ученые считают нейробиологию важным источником идей, другие не упоминают ее вовсе. Важно отметить, что попытки понять, как работает мозг на алгоритмическом уровне, не прекращаются. Эта область исследований, известная под названием «вычислительная нейробиология», не совпадает с глубоким обучением. Нередко ученые переходят из одной области в другую. Предмет глубокого обучения – построение компьютерных систем, способных успешно решать задачи, требующие интеллекта, а предмет вычислительной нейробиологии – построение более точных моделей работы мозга.

## 1.1 Определения

**Сверточная сеть** (LeCun, 1989), она же сверточная нейронная сеть (СНС), – это специальный вид нейронной сети для обработки данных с сеточной топологией. Примерами могут служить временные ряды, которые можно рассматривать как одномерную сетку примеров, выбираемых через регулярные промежутки времени, а также изображения, рассматриваемые как двумерная сетка пикселей. Сверточные сети добились колоссального успеха в практических приложениях. Своим названием они обязаны использованию математической операции свертки. Свертка – это особый вид линейной операции. Сверточные сети – это просто нейронные сети, в которых вместо общей операции умножения на матрицу, по крайней мере в одном слое, используется свертка.

Обычно операция, используемая в сверточной нейронной сети, не вполне соответствует определению свертки в других областях, например, в инженерных

дисциплинах и в чистой математике. Опишем несколько вариантов функции свертки, широко применяемых в нейронных сетях. Также покажем, как можно применить свертку к различным видам данных в пространствах разной размерности. Затем обсудим, как повысить эффективность свертки. Сверточные сети – яркий пример того, как принципы нейробиологии оказывают влияние на глубокое обучение. Цель главы – описать инструментарий, предоставляемый сверточными сетями. Прогресс в изучении сверточных сетей настолько быстрый, что сообщения о новой оптимальной архитектуре для данного эталонного теста появляются через каждые несколько недель или месяцев, поэтому называть какую-то архитектуру лучшей всегда преждевременно. Тем не менее все лучшие архитектуры скомпонованы из описанных далее блоков.

## Свёртка

В самом общем виде свертка – это операция над двумя функциями вещественного аргумента. Чтобы обосновать определение свертки, начнем с примеров возможных функций. Допустим, что мы следим за положением космического корабля с помощью лазерного датчика. Наш датчик выдает единственное значение  $x(t)$ , положение корабля. Операция свертки в момент  $t$ . Переменные  $x$  и  $t$  принимают вещественные значения, т. е. показания датчика в любые два момента времени могут различаться.

Теперь предположим, что датчик подвержен помехам. Чтобы получить менее зашумленную оценку положения корабля, необходимо усреднить несколько результатов измерений. Разумеется, недавние измерения более важны, поэтому мы хотим вычислять взвешенное среднее, придавая недавним измерениям больший вес. Для этого можно воспользоваться весовой функцией  $w(a)$ , где  $a$  – давность измерения. Применив такую операцию усреднения в каждый момент времени, мы получим новую функцию, которая дает сглаженную оценку положения космического корабля:

$$s(t) = \int x(a)w(t-a)da.$$

Эта операция называется сверткой и обычно обозначается звездочкой:

$$s(t) = (x * w)(t).$$

В этом примере  $w$  должна быть функцией плотности вероятности, иначе усреднения не получится. Кроме того,  $w$  должна быть равна 0 для всех отрицательных значений аргумента, иначе она будет способна заглядывать в будущее, что вряд ли в пределах наших возможностей. Но эти ограничения характерны только для данного примера. В общем случае свертку можно определить для любых функций, для которых определен показанный выше интеграл, и использовать не только для получения взвешенного среднего.

В терминологии сверточных сетей первый аргумент (в нашем примере функция  $x$ ) называется **входом**, а второй (функция  $w$ ) – **ядром**. Выход иногда называют **картой признаков**.

Лазерных датчиков, способных выдавать результаты измерений в любой момент времени, в действительности не бывает. Обычно при работе с данными в компьютере время дискретизировано, и наш датчик будет выдавать данные через регулярные интервалы. Пожалуй, было бы реалистичнее предположить, что лазер производит измерения раз в секунду. Индекс момента времени  $t$  может принимать только целые значения. Если теперь предположить, что  $x$  и  $w$  определены только для целых  $t$ , то мы получим определение дискретной свертки:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a).$$

В приложениях машинного обучения входом обычно является многомерный массив данных, а ядром – многомерный массив параметров, адаптированных алгоритмом обучения. Будем называть эти массивы **тензорами**. Поскольку каждый элемент входа и ядра должен храниться отдельно в явном виде, обычно предполагается, что эти функции равны нулю всюду, кроме конечного множества точек, для которых мы храним значения.

На практике это означает, что сумму от минус до плюс бесконечности можно заменить суммированием по конечному числу элементов массива.

Наконец, мы часто используем свертки сразу по нескольким осям. Например, если входом является двумерное изображение  $I$ , то и ядро должно быть двумерным:

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n).$$

Операция свертки коммутативна, поэтому формулу можно записать и так:

$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n).$$

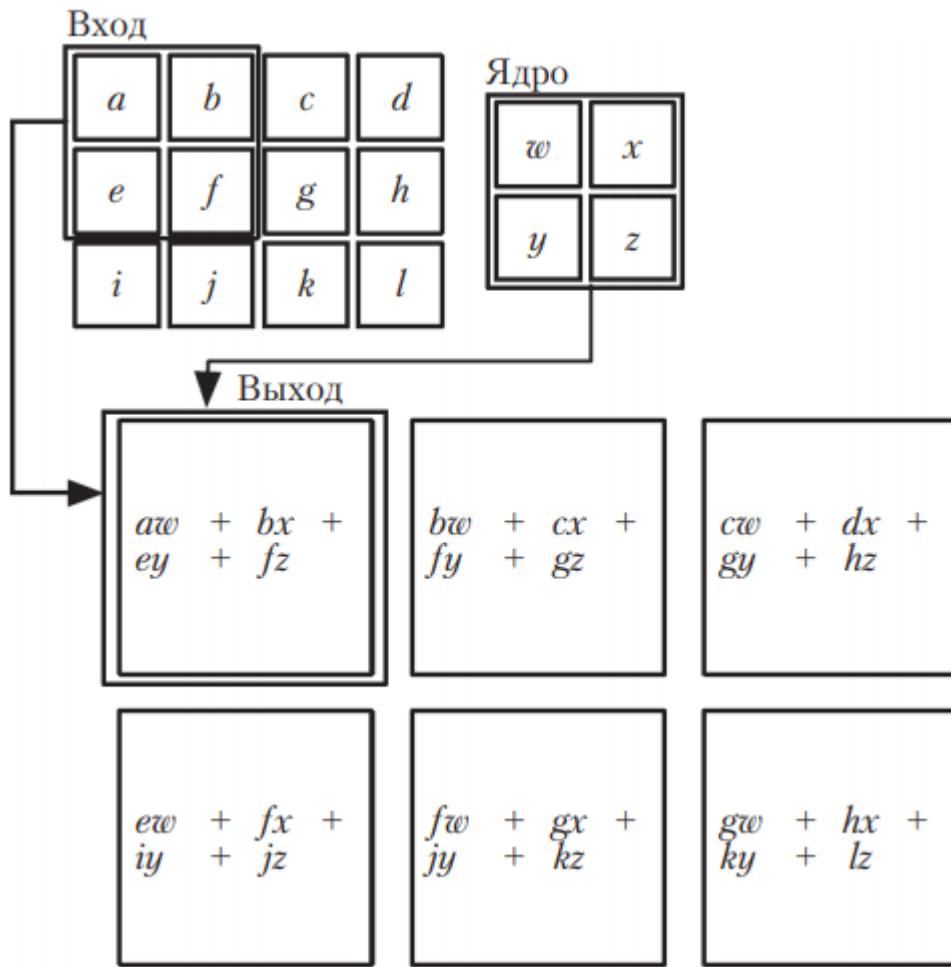
Обычно вторую формулу проще реализовать в библиотеке машинного обучения, поскольку диапазон допустимых значений  $m$  и  $n$  меньше. Свойство коммутативности свертки имеет место, потому что мы **отразили** ядро относительно входа, т. е. при увеличении  $m$  индекс входа увеличивается, а индекс ядра уменьшается. Единственная причина такого отражения – обеспечить коммутативность. И хотя коммутативность полезна для доказательства теорем, в реализации нейронных сетей она обычно роли не играет. Вместо этого во многих библиотеках реализована родственная функция – **перекрестная корреляция** – та же свертка, только без отражения ядра:

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n).$$

Во многих библиотеках машинного обучения реализована именно перекрестная корреляция, но называется она сверткой. Далее мы будем придерживаться этого соглашения и называть сверткой обе операции, а в тех местах, где отражение ядра имеет значение, будем это явно оговаривать. В контексте машинного обучения алгоритм обучения ставит найденные значения ядра в правильную позицию, поэтому если алгоритм основан на свертке с отражением ядра, то он обучит ядро, отраженное относительно того, которое обучено алгоритмом со сверткой без отражения. Редко бывает так, чтобы свертка использовалась в машинном обучении сама по себе; чаще она

комбинируется с другими функциями, и такие комбинации не коммутативны вне зависимости от того, используется в ядре-свертке отражение или нет. На Рисунке 2 приведен пример свертки (без отражения ядра) в применении к двумерному тензору.

Дискретную свертку можно рассматривать как умножение на матрицу, на элементы которой наложены некоторые ограничения. Например, в случае одномерной дискретной свертки каждая строка матрицы должна быть равна предыдущей, сдвинутой на один элемент. Это утверждение называется **теоремой Теплица**. В двумерном случае свертке соответствует **дважды блочно-циркулянтная матрица**. Помимо ограничений на равенство некоторых элементов, свертке обычно соответствует сильно разреженная матрица (в которой большинство элементов равно нулю). Связано это с тем, что ядро, как правило, гораздо меньше входного изображения. Любой алгоритм нейронной сети, основанный на умножении матриц и не зависящий от особенностей структуры этих матриц, должен работать и со сверткой без каких-либо модификаций самой сети. В типичных сверточных нейронных сетях все же используются особенности структуры, чтобы организовать эффективную обработку больших входов, но с теоретической точки зрения это необязательно.



*Рисунок 2. Пример двумерной свертки без отражения ядра*

Со сверткой связаны три важные идеи, которые помогают улучшить систему машинного обучения: **разреженные взаимодействия**, **разделение параметров** и **эквивариантные представления**. Кроме того, свертка предоставляет средства для работы со входами переменного размера.

В слоях традиционной нейронной сети применяется умножение на матрицу параметров, в которой взаимодействие между каждым входным и каждым выходным блоками описывается отдельным параметром. Это означает, что каждый выходной блок взаимодействует с каждым входным блоком. Напротив, в сверточных сетях взаимодействия обычно разреженные (это свойство называют еще **разреженной**

**связностью**, или разреженными весами). Достигается это за счет того, что ядро меньше входа. Например, входное изображение может содержать тысячи или миллионы пикселей, но небольшие значимые признаки, например, границы, можно обнаружить с помощью ядра, охватывающего всего десятки или сотни пикселей. Следовательно, нужно хранить меньше параметров, а это снижает требования модели к объему памяти и повышает ее статистическую эффективность. Кроме того, для вычисления выхода потребуется меньше операций. Все вместе обычно намного повышает эффективность сети. Если имеется  $m$  входов и  $n$  выходов, то для умножения матриц нужно  $m * n$  параметров, и сложность практически используемых алгоритмов составляет  $O(m * n)$  (в расчете на один пример). Если ограничить число соединений с каждым выходом величиной  $k$ , то потребуется только  $k * n$  параметров, и сложность составит  $O(k * n)$ . Во многих практических приложениях можно получить хорошее качество на задаче машинного обучения, когда  $k$  на несколько порядков меньше  $m$ . В глубокой сверточной сети блоки нижних уровней могут косвенно взаимодействовать с большей частью сети. Это дает сети возможность эффективно описывать сложные взаимодействия между многими переменными путем составления из простых строительных блоков, каждый из которых описывает только разреженные взаимодействия.

Под **разделением параметров** понимают, что один и тот же параметр используется в нескольких функциях модели. В традиционной нейронной сети каждый элемент матрицы весов используется ровно один раз при вычислении выхода слоя. Он умножает на один элемент входа, и больше мы к нему никогда не возвращаемся. Вместо употребления термина «разделение параметров» можно сказать, что в сети присутствуют **связанные веса**, поскольку значение веса, примененного к одному входу, связано со значением веса, примененного где-то еще. В сверточной нейронной сети каждый элемент ядра применяется к каждой позиции входа (за исключением, быть может, некоторых граничных пикселей – в зависимости от того, как решено обрабатывать границу). Разделение параметров означает, что вместо обучения отдельного набора параметров для каждой точки мы должны обучить только один набор. Это не влияет на время прямого

распространения – оно по-прежнему имеет порядок  $O(k * n)$ , – но дополнительно уменьшает требования к объему памяти: достаточно хранить  $k$  параметров. Напомним, что  $k$  обычно на несколько порядков меньше  $m$ . Поскольку величины  $m$  и  $n$  приблизительно равны, то  $k$  практически несущественно по сравнению с  $m * n$ . Таким образом, свертка многократно эффективнее умножения матриц с точки зрения требований к памяти и статистической эффективности. Разреженная связность и разделение параметров кардинально улучшают эффективность линейной функции при обнаружении границ в изображении.

В случае свертки специальный вид разделения параметров наделяет слой свойством, которое называется **эквивариантностью** относительно параллельного переноса. Говорят, что функция эквивариантна, если при изменении входа выход меняется точно так же. Точнее, функция  $f(x)$  эквивариантна относительно функции  $g$ , если  $f(g(x)) = g(f(x))$ . В случае свертки, если  $g$  – параллельный перенос, или сдвиг входа, то функция свертки эквивариантна относительно  $g$ . Пусть, например, функция  $I$  определяет яркость в точках с целыми координатами, и пусть  $g$  – функция, отображающая одну функцию изображения в другую функцию изображения, так что  $I' = g(I)$  – функция изображения, для которой  $I'(x, y) = I(x - 1, y)$ . Это сдвиг каждого пикселя  $I$  на одну позицию вправо. Если применить это преобразование к  $I$ , а затем выполнить свертку, то результат будет таким же, как если бы мы сначала применили свертку к  $I'$ , а затем преобразование  $g$  к результату. При обработке временных рядов это означает, что свертка порождает своего рода временную шкалу, на которой показано время появления различных признаков во входных данных.

Если перенести событие на более поздний момент времени во входных данных, то на выходе появится точно такое же его представление, только позже. А при работе с изображениями свертка создает двумерную карту появления определенных признаков во входном изображении. Если переместить объект во входном изображении, то его представление на выходе переместится на такую же величину. Это бывает нужно, когда мы знаем, что некоторая функция от небольшого числа пикселей полезна при применении

к нескольким участкам входа. Например, в случае обработки изображений полезно обнаруживать границы в первом слое сверточной сети. Одни и те же границы встречаются более-менее везде в изображении, поэтому имеет смысл разделять параметры по всему изображению. Но в некоторых случаях такое глобальное разделение параметров нежелательно. Например, если мы обрабатываем изображения, которые были кадрированы, так чтобы в центре оказалось лицо человека, то, наверное, хотим выделять разные признаки в разных точках – часть сети будет обрабатывать верхнюю часть лица в поисках бровей, а другая часть – искать подбородок в нижней части лица.

Свертка не эквивариантна относительно некоторых других преобразований, например, масштабирования или поворота. Для обработки таких преобразований нужны другие механизмы. Наконец, существуют типы данных, которые нельзя обработать с помощью нейронных сетей, определяемых путем умножения на матрицу фиксированной формы. Свертка позволяет обрабатывать некоторые данные такого рода.

## Скрытые блоки

Блоки линейной ректификации – отличный выбор для скрытых блоков в отсутствие дополнительных аргументов. Но есть и много других типов. Бывает трудно решить, какой тип взять в конкретном случае (хотя обычно блоки линейной ректификации дают приемлемый результат). Процесс проектирования – это последовательность проб и ошибок, когда высказывается гипотеза о подходящем блоке, затем обучается сеть с таким типом скрытых блоков и результат оценивается на контрольном наборе.

Некоторые скрытые блоки, включенные в список, не являются всюду дифференцируемыми. Например, функция линейной ректификации  $g(z) = \max(0, z)$  не дифференцируема в точке  $z = 0$ . Может показаться, что из-за этого  $g$  непригодна для работы с алгоритмом обучения градиентными методами. Но на практике градиентный спуск работает для таких моделей машинного обучения достаточно хорошо. Отчасти это связано с тем, что алгоритмы обучения нейронных сетей обычно не достигают локального

минимума функции стоимости, а просто находят достаточно малое значение. Поскольку мы не ожидаем, что обучение выйдет на точку, где градиент равен 0, то можно смириться с тем, что минимум функции стоимости соответствует точкам, в которых градиент не определен. Недифференцируемые скрытые блоки обычно не дифференцируемы лишь в немногих точках. В общем случае функция  $g(z)$  имеет производную слева, определяемую коэффициентом наклона функции слева от  $z$ , и аналогично производную справа. Функция дифференцируема в точке  $z$ , только если производные слева и справа определены и равны между собой. Для функций, встречающихся в контексте нейронных сетей, обычно определены производные слева и справа. Для функции  $g(z) = \max(0, z)$  производная слева в точке  $z = 0$  равна 0, а производная справа – 1. В программных реализациях обучения нейронной сети обычно возвращается какая-то односторонняя производная, а не сообщается, что производная не определена и не возбуждается исключение. Эвристически это можно оправдать, заметив, что градиентная оптимизация на цифровом компьютере в любом случае подвержена численным погрешностям. Когда мы просим вычислить  $g(0)$ , крайне маловероятно, что истинное значение действительно равно 0. Скорее всего, это какое-то малое значение  $\epsilon$ , округленное до 0. В некоторых случаях возможны теоретически более убедительные обоснования, но обычно к обучению нейронных сетей они не относятся. Важно, что на практике можно спокойно игнорировать недифференцируемость функций активации скрытых блоков. В большинстве случаев скрытый блок можно описать следующим образом: получить вектор входов  $x$ , вычислить аффинное преобразование  $z = W^T x + b$  и применить к каждому элементу нелинейную функцию  $g(z)$ . Друг от друга скрытые блоки отличаются только функцией активации  $g(z)$ .

Блоки линейной ректификации обычно применяются после аффинного преобразования, т.е.  $h = g(W^T x + b)$ .

При инициализации параметров аффинного преобразования рекомендуется присваивать всем элементам  $b$  небольшое положительное значение, например 0.1. Тогда блок линейной ректификации в начальный момент с большой вероятностью окажется

активен для большинства обучающих примеров, и производная будет отлична от нуля. Существует несколько обобщений блока линейной ректификации. Большинство из них демонстрирует более высокое качество лишь в отдельных случаях.

Недостатком блоков линейной ректификации является невозможность обучить их градиентными методами на примерах, для которых функция активации блока равна нулю. Различные обобщения гарантируют, что градиент имеется в любой точке. Три обобщения блоков линейной ректификации основаны на использовании ненулевого углового коэффициента  $\alpha_i$ , когда  $z_i < 0$ ;  $h_i = g(z, \alpha)_i = \max(0, z_i) + \alpha_i \min(0, z_i)$ . В случае абсолютной ректификации (absolute value rectification) берутся фиксированные значения  $\alpha_i = -1$ , так что  $g(z) = |z|$ . Такая функция активации используется при распознавании объектов в изображении (Jarrett et al., 2009), где имеет смысл искать признаки, инвариантные относительно изменения полярности освещения. Другие обобщения находят более широкие применения. В случае **ReLU с утечкой** (leaky ReLU) (Maas et al., 2013)  $\alpha_i$  принимаются равными фиксированному малому значению, например 0.01, а в случае **параметрического ReLU**, или **PReLU**  $\alpha_i$ , считается обучаемым параметром (He et al., 2015).

**Maxout-блоки** (Goodfellow et al., 2013a) – это дальнейшее обобщение блоков линейной ректификации. Вместо того чтобы применять функцию  $g(z)$  к каждому элементу, вектор  $z$  разбивается на группы по  $k$  значений. Затем каждый maxout-блок выводит максимальный элемент одной из групп.

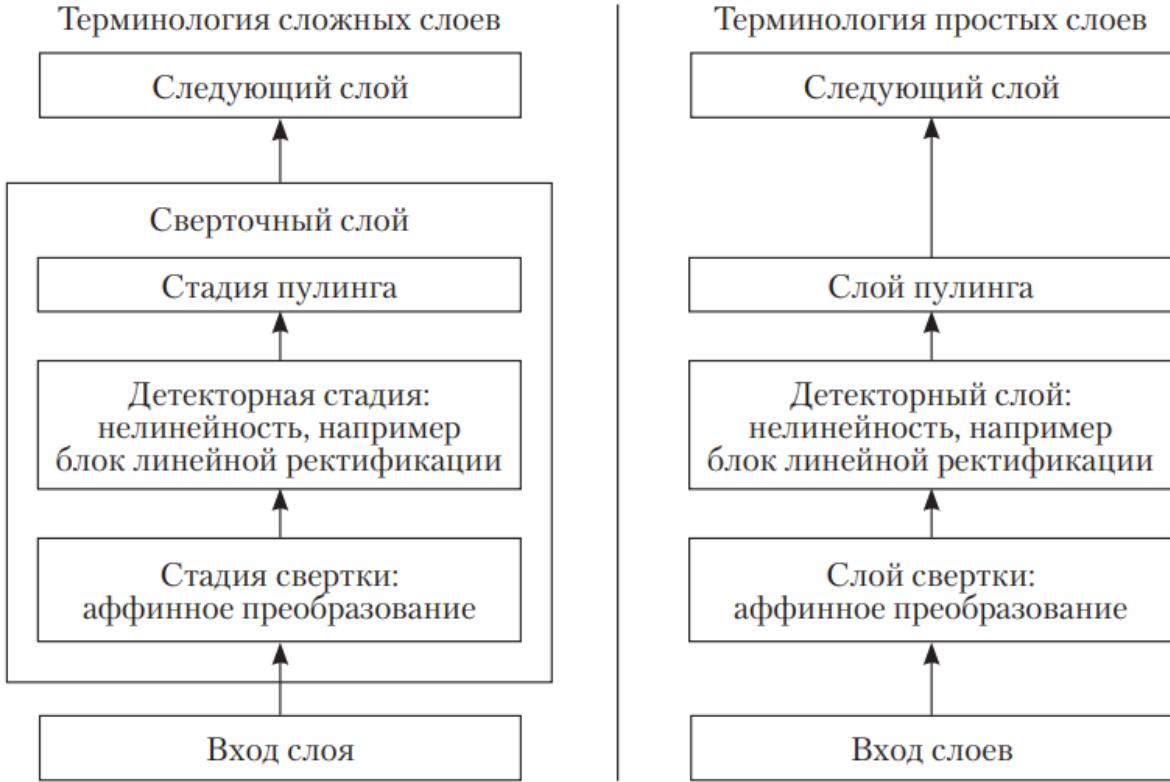
Блоки линейной ректификации стали использоваться сравнительно недавно, а раньше в большинстве нейронных сетей в роли функции активации применялась логистическая сигмоида  $g(z) = \sigma(z)$  или гиперболический тангенс  $g(z) = \tanh(z)$ .

Эти функции активации тесно связаны:  $\tanh(z) = 2\sigma(2z) - 1$ . В отличие от кусочно-линейных, сигмоидальные блоки близки к асимптоте в большей части своей области определения – приближаются к высокому значению, когда  $z$  стремится к бесконечности, и к низкому, когда  $z$  стремится к минус бесконечности. Высокой чувствительностью они обладают только в окрестности нуля. Из-за насыщения

сигмоидальных блоков градиентное обучение сильно затруднено. Поэтому использование их в качестве скрытых блоков в сетях прямого распространения ныне не рекомендуется. Применение же в качестве выходных блоков совместимо с обучением градиентными методами, если функция стоимости компенсируется насыщением сигмоиды в выходном слое. Если использовать сигмоидальную функцию активации необходимо, то лучше взять не логистическую сигмоиду, а гиперболический тангенс. Он ближе к тождественной функции в том смысле, что  $\tanh(0) = 0$ , тогда как  $\sigma(0) = 1/2$ . Поскольку  $\tanh$  походит на тождественную функцию в окрестности нуля, обучение глубокой нейронной сети  $y = w^T \tanh(U^T \tanh(V^T x))$  напоминает обучение линейной модели  $y = w^T U^T V^T x$ , при условии что сигналы активации сети удается удерживать на низком уровне. При этом обучение сети с функцией активации  $\tanh$  упрощается. Сигмоидальные функции активации все же применяются, но не в сетях прямого распространения. К рекуррентным сетям, многим вероятностным моделям и некоторым автокодировщикам предъявляются дополнительные требования, исключающие использование кусочно-линейных функций активации и делающие сигмоидальные блоки более подходящими, несмотря на проблемы насыщения.

## Пуллинг

Типичный слой сверточной сети состоит из трех стадий Рисунок 3. На первой стадии слой параллельно выполняет несколько сверток и порождает множество линейных активаций. На второй стадии каждая линейная активация пропускается через нелинейную функцию активации, например, функцию линейной ректификации. Этую стадию часто называют детекторной. На третьей стадии используется функция пулинга для дальнейшей модификации выхода слоя.



**Рисунок 3. Компоненты типичного слоя свёрточной нейронной сети. Двойственная терминология. (Слева) Свёрточная сеть небольшой набор сложных многостадийных слоёв. (Справа) Свёрточная сеть большой набор простых слоёв.**

Функция пулинга заменяет выход сети в некоторой точке сводной статистикой близлежащих выходов. Например, операция **max-пулинга** (Zhou and Chellappa, 1988) возвращает максимальный выход в прямоугольной окрестности. Из других употребительных функций пулинга отметим усреднение по прямоугольной окрестности, L2 -норму в прямоугольной окрестности и взвешенное среднее с весами, зависящими от расстояния до центрального пикселя. В любом случае пулинг позволяет сделать представление приблизительно инвариантным относительно малых параллельных переносов входа. Инвариантность относительно параллельного переноса означает, что если сдвинуть вход на небольшую величину, то значения большинства подвергнутых пулингу выходов не изменятся. **Локальная инвариантность относительно параллельного переноса полезна, если нас больше интересует сам факт существования некоторого признака, а не его точное местонахождение.** Например, когда мы хотим определить, присутствует ли в изображении лицо, нам не важно

положение глаз с точностью до пикселя, нужно только знать, есть ли глаз слева и глаз справа. В других ситуациях важнее сохранить местоположение признака. Например, если мы ищем угловую точку, образованную пересечением двух границ, ориентированных определенным образом, то необходимо сохранить положение границ настолько точно, чтобы можно было проверить, пересекаются ли они.

Пулинг можно рассматривать как добавление бесконечно сильного априорного предположения, что обучаемая слоем функция должна быть инвариантна к малым параллельным переносам. Если это предположение правильно, то оно может существенно улучшить статистическую эффективность сети. Пулинг по пространственным областям порождает инвариантность к параллельным переносам, но если он производится по выходам сверток с различными параметрами, то признаки могут обучаться, к каким преобразованиям стать инвариантными.

Поскольку пулинг агрегирует отклики по целой окрестности, количество блоков пулинга можно сделать меньшим, чем количество детекторных блоков, если агрегировать статистику по областям, отстоящим друг от друга на  $k > 1$  пикселей. Тем самым повышается вычислительная эффективность сети, поскольку следующему слою предстоит обработать примерно в  $k$  раз меньше входов. Если число параметров в следующем слое – функция от размера входа (например, когда следующий слой полно связанный и основан на умножении матриц), то уменьшение размера входа также может повысить статистическую эффективность и уменьшить требования к объему памяти для хранения параметров.

В большинстве задач пулинг необходим для обработки входов переменного размера. Например, если мы хотим классифицировать изображения разного размера, то код слоя классификации должен иметь фиксированный размер. Обычно это достигается за счет варьирования величины шага между областями пулинга, так чтобы слой классификации всегда получал одинаковый объем сводной статистики независимо от размера входа. Так, можно определить финальный слой пулинга в сети, так чтобы он

выводил четыре сводных статистических показателя, по одному на каждый квадрант изображения, вне зависимости от размера самого изображения.

Существуют кое-какие теоретические рекомендации по выбору вида пулинга в различных ситуациях (Boureau et al., 2010). Можно также динамически агрегировать признаки, например, путем выполнения алгоритма кластеризации в местах интересных признаков (Boureau et al., 2011). При таком подходе получаются различные множества областей пулинга для каждого изображения. Другой подход – обучить единую структуру пулинга и затем применять ее ко всем изображениям (Jia et al., 2012). Пулинг может внести усложнения в некоторые архитектуры нейронных сетей, где используется исходящая информация, как, например, машины Больцмана и автокодировщики.

## **1.2 Современные методы супер-разрешения**

### **Интерполяция методом ближайшего соседа**

**Интерполяция методом ближайшего соседа** (ступенчатая интерполяция) — метод интерполяции, при котором в качестве промежуточного значения выбирается ближайшее известное значение функции. Интерполяция методом ближайшего соседа является самым простым методом интерполяции.

### **Бикубическая интерполяция**

**Бикубическая интерполяция** — в вычислительной математике расширение кубической интерполяции на случай функции двух переменных, значения которой заданы на двумерной регулярной сетке. Поверхность, полученная в результате бикубической интерполяции является гладкой функцией на границах соседних квадратов, в отличие от поверхностей, полученных в результате билинейной интерполяции или интерполяции методом ближайшего соседа.

Бикубическая интерполяция часто используется в обработке изображений, давая более качественное изображение по сравнению с билинейной интерполяцией. Также бикубическая интерполяция применяется в алгоритмах управления станков с ЧПУ для учета неровностей плоскостей, например, при фрезеровке печатных плат.

## **Методы на основе разреженного кодирования**

**Разреженное словарное обучение** - это метод обучения представлению, целью которого является нахождение разреженного представления входных данных (также известного как разреженное кодирование) в форме линейной комбинации базовых элементов, а также самих этих базовых элементов. Эти элементы называются атомами, и они составляют словарь. Атомы в словаре не обязательно должны быть ортогональными, и они могут быть переполненным охватывающим набором. Эта проблемная установка также позволяет размерности представляемых сигналов быть выше, чем один из наблюдаемых сигналов. Два вышеупомянутых свойства приводят к наличию, казалось бы, избыточных атомов, которые допускают множественные представления одного и того же сигнала, но также обеспечивают улучшение разреженности и гибкости представления.

Одним из наиболее важных применений редкого изучения словаря является область сжатого восприятия или восстановления сигнала. При сжатии считывания сигнал высокой размерности может быть восстановлен только с помощью нескольких линейных измерений при условии, что сигнал является разреженным или почти разреженным. Поскольку не все сигналы удовлетворяют этому условию разреженности, очень важно найти разреженное представление этого сигнала, такое как вейвлет-преобразование или направленный градиент растеризованной матрицы. Как только матрица или вектор высокой размерности передаются в разреженное пространство, для восстановления сигнала могут использоваться различные алгоритмы восстановления, такие как базовое преследование, CoSaMP или быстрые не итерационные алгоритмы.

Одним из ключевых принципов изучения словаря является то, что словарь должен быть выведен из входных данных. Появление методов изучения разреженных словарей

было стимулировано тем фактом, что при обработке сигналов обычно требуется представлять входные данные, используя как можно меньше компонентов. До этого подхода общей практикой было использование предопределенных словарей (таких как преобразования Фурье или вейвлет). Тем не менее, в некоторых случаях словарь, который подготовлен для соответствия входным данным, может значительно улучшить разреженность, который имеет приложения для декомпозиции, сжатия и анализа данных и используется в областях шумоподавления и классификации изображений, обработки видео и аудио. Редкие и переполненные словари имеют огромное применение в сжатии изображений, объединении изображений и рисовании.

### **Алгоритмы:**

1. **Метод оптимальных направлений** (Method of optimal directions или MOD) был одним из первых методов, введенных для решения проблемы разреженного изучения словаря. Основная идея заключается в том, чтобы решить проблему минимизации с учетом ограниченного числа ненулевых компонентов вектора представления. MOD оказался очень эффективным методом для низко размерных входных данных, требующим всего нескольких итераций для сходжения. Однако из-за высокой сложности операции обращения матриц вычисление псевдообращения в многомерных случаях во многих случаях неразрешимо.
2. **K-SVD** - это алгоритм изучения словаря для создания словаря для разреженных представлений с помощью подхода разложения по сингулярным числам. K-SVD является обобщением метода кластеризации k-средних, и он работает путем итеративного чередования разреженного кодирования входных данных на основе текущего словаря и обновления атомов в словаре для лучшего соответствия данным. K-SVD широко используется в таких приложениях, как обработка изображений, обработка аудио, биология и анализ документов.
3. Можно также применить широко распространенный метод **стохастического градиентного спуска** с итерационной проекцией для решения этой проблемы. Идея этого метода состоит в том, чтобы обновить словарь с использованием

стохастического градиента первого порядка и спроектировать его на набор ограничений.

4. Алгоритм, основанный на решении **двойной задачи Лагранжа**, обеспечивает эффективный способ поиска для словаря, не имеющего сложностей, вызванных функцией разреженности.
5. **LASSO** (Least Absolute Shrinkage and Selection Operator или оператора наименьшей абсолютной усадки и выбора) - это метод регрессионного анализа, который выполняет как выбор переменных, так и регуляризацию, чтобы повысить точность прогнозирования и интерпретируемость создаваемой им статистической модели. Первоначально он был введен в литературу по геофизике в 1986 году, а затем независимо переоткрыт и популяризирован в 1996 году Робертом Тибштади, который придумал этот термин и дал дополнительную информацию о наблюдаемых характеристиках.

Изначально Лассо было сформулировано для моделей наименьших квадратов, и этот простой случай раскрывает существенную информацию о поведении оценщика, включая его отношение к регрессии гребня и выбору наилучшего подмножества, а также связь между оценками коэффициента лассо и так называемой мягкой установкой порога. Это также показывает, что (подобно стандартной линейной регрессии) оценки коэффициентов не обязательно должны быть уникальными, если особенности коллинеарные.

Несмотря на то, что первоначально метод был определен для наименьших квадратов, лассо-регуляризация легко распространяется на широкий спектр статистических моделей, включая обобщенные линейные модели, обобщенные уравнения оценки, модели пропорциональных рисков и М-оценки, простым способом. Способность Лассо выполнять выбор подмножеств зависит от формы ограничения и имеет множество интерпретаций, в том числе с точки зрения геометрии, байесовской статистики и анализа выпуклых функций.

## **2. Супер-разрешения с помощью свёрточных нейронных сетей**

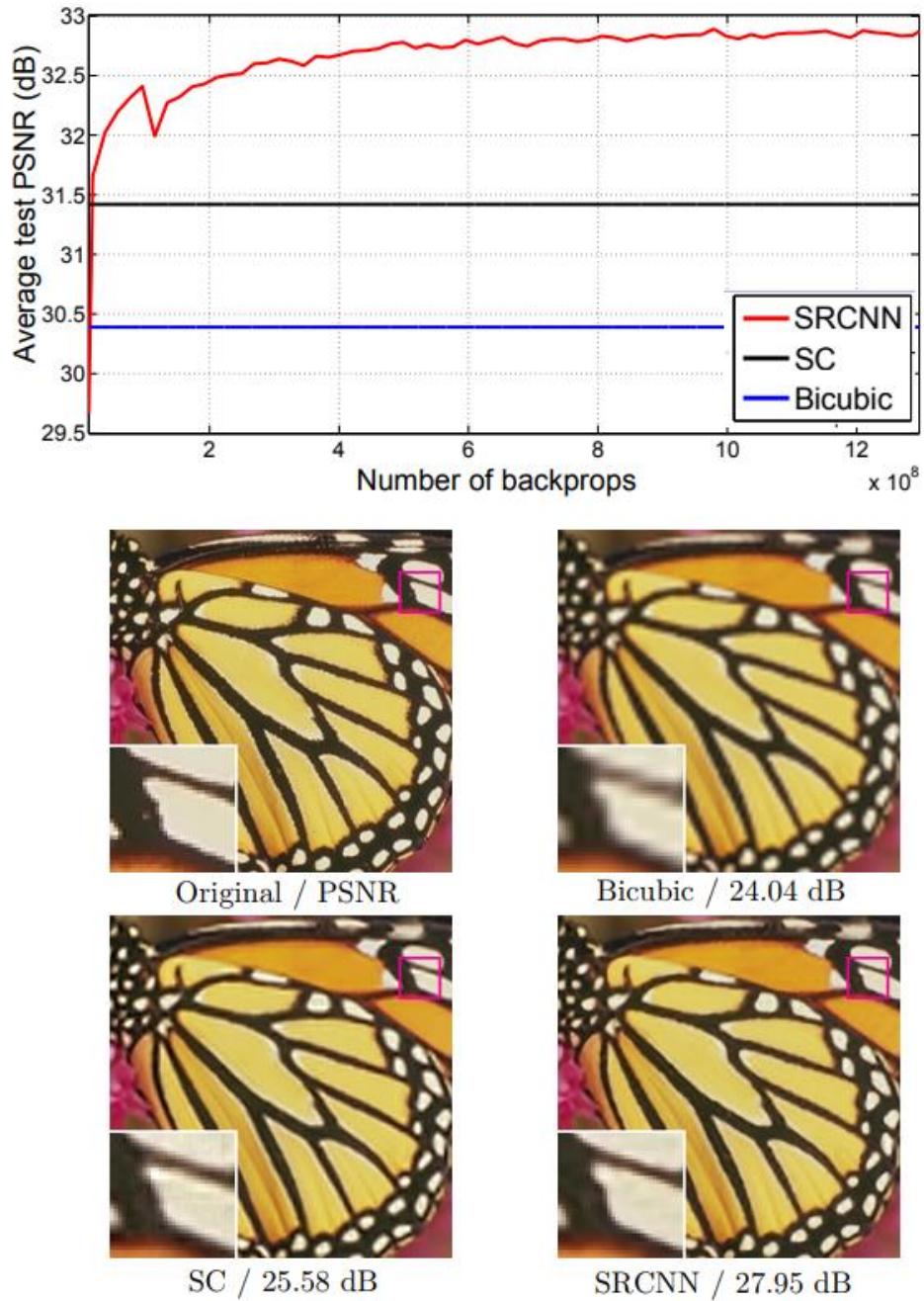
### **2.1 Введение**

Супер-разрешение изображения – это восстановленное изображение более высокого разрешения изображения, чем исходное изображение. Проблема конструирования таких изображений одна из классических в компьютерном зрении. Эта проблема изначально некорректна, так как существует множество решений данной задачи. Другими словами, это недоопределенная обратная задача, решение которой не является уникальным. Эта проблема обычно смягчается путем ограничения пространства решения сильной априорной информацией. Последние современные методы в основном используют стратегию, основанную на примерах. Эти методы либо используют внутренние сходства одного и того же изображения, либо изучают функции отображения из низкого разрешения в высокое.

Метод на основе разреженного кодирования является одним из самых репрезентативных методов Super-resolution, основанный на внешних примерах. Этот метод включает несколько шагов. Во-первых, пересекающиеся патчи плотно (с маленьким шагом) вырезаются из исходного изображения и предобрабатываются (например, вычитается среднее и нормализуются). Эти патчи затем кодируются словарём более низкого разрешения. Разреженные коэффициенты передаются в словарь с высоким разрешением для восстановления патчей более высокого разрешения. Перекрывающиеся фрагменты агрегируют (например, взвешенное среднее) для получения окончательного результата. Такая последовательность действий используются большинством методов, основанных на примерах, которые выделяют особое внимание оптимизации словарей или созданию эффективных функций отображения, однако остальные шаги таких подходов редко можно оптимизировать.

В данной работе хотелось показать, что такой сложный алгоритм из нескольких шагов эквивалентен свёрточной нейронной сети. Принимая это за факт, рассмотрим сеть, которая непосредственно изучает прямое отображение между изображениями с низким и высоким разрешениями. Этот метод принципиально отличается от существующих подходов, основанных на примерах тем, что он не явно изучает словари или многообразие для моделирования пространства патчей. Это неявно достигается через скрытые слои нейронной сети. Кроме того, извлечение и агрегация также сформулированы в этих же слоях, поэтому они и участвуют в оптимизации. В этом методе полный алгоритм Super-resolution полностью получен с помощью обучения с пред- и постобработкой.

Авторы этого метода назвали его Super-Resolution Convolutional Neural Network (SRCNN). Предлагаемый SRCNN обладает некоторыми привлекательными особенностями. Во-первых, его структура специально спроектирована с учетом простоты и обеспечения превосходной точности по сравнению с современными методами, основанными на примерах. На Рисунке 1 показано сравнение с такими методами.



*Рисунок 1. Сравнение SRCNN с популярными алгоритмами*

Во-вторых, при умеренном количеством фильтров и слоёв в сети, этот метод обеспечивает высокую скорость для практического использования онлайн даже на CPU. Он работает быстрее чем алгоритмы, основанные на примерах, так как он полностью прямонаправленный и не нуждается в решении каких-либо оптимизационных проблем.

В-третьих, примеры показывают, что восстановление сети может быть улучшено, когда доступны большие и разнообразные датасеты и используется более глубокая модель. Что напротив для других методов, основанных на примерах, может представлять проблемы. Также следует отметить, что нейронная сеть может обрабатывать сразу три цветовых канала изображения для достижения лучших результатов суперразрешения.

В целом, вклад этого подхода можно описать в трех аспектах:

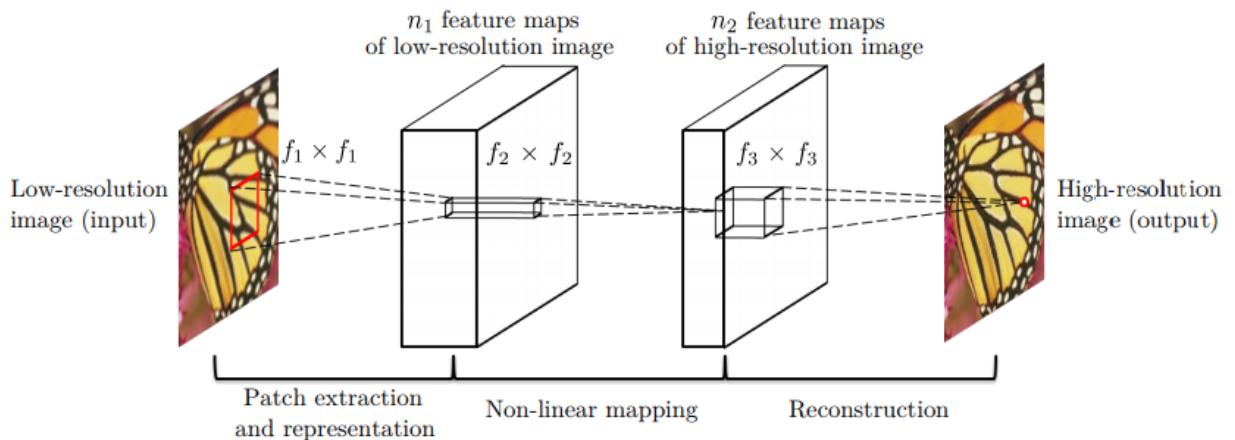
1. Представлена полностью свёрточная нейронная сеть для Super-resolution изображения. Сеть непосредственно обучается отображать изображение низкого разрешения в изображение высокого разрешения с минимальной пред- и постобработкой.
2. Установлена связь между Deep Learning Super-Resolution методом и традиционными методами, основанными на разреженном кодировании. Эта связь обеспечивает руководство к проектированию структуры сети.
3. Продемонстрировано, что глубокое обучение применимо в классической проблеме компьютерного зрения суперразрешения, и показывает отличные результаты качества и скорости.

## **2.2 Описание метода**

Рассматривая одно изображение низкого разрешения, во-первых, увеличиваем разрешение до желаемого размера используя бикубическую интерполяцию, это единственная предобработка. Назовём интерполированное изображение как  $\mathbf{Y}$ . Задача состоит в том, чтобы из изображения  $\mathbf{Y}$  восстановить изображение  $F(\mathbf{Y})$ , которое максимально похоже на изображение  $\mathbf{X}$  с высоким разрешением. Для простоты  $\mathbf{Y}$  все равно воспринимается как изображение низкого разрешения, хотя его разрешение такое же, как и у  $\mathbf{X}$ . Метод сводится к обучению отображения  $F$ , которое концептуально состоит из трех операций:

- Извлечение и представление патчей:** эта операция извлекает пересекающиеся патчи из изображения  $\mathbf{Y}$  низкого разрешения и представляет каждый патч как вектор высокой размерности. Эти векторы составляют набор карт признаков, число которых равно размерности векторов.
- Нелинейное отображение:** эта операция нелинейно отображает каждый многомерный вектор в другой многомерный вектор. Каждый отображенный вектор является представлением патча высокого разрешения. Эти векторы составляют другой набор признаков.
- Реконструкция:** эта операция агрегирует вышеприведенные представления патчей высокого разрешения для генерации окончательного изображения высокого разрешения.

Все эти операции образуют одну свёрточную нейронную сеть. Обзор сети изображен на Рисунке 2. Далее более подробно разберем каждую операцию.



*Рисунок 2. Структура SRCNN*

## Извлечение и представление патчей

Это популярная стратегия в восстановлении изображений состоит в том, чтобы плотно извлекать патчи и затем представлять их с помощью предварительно обученного

базиса, как PCA, DCT, Haar и т.д. Это эквивалентно свёртке изображения с множеством фильтров, каждый из которых является базисом. В этой формулировке в оптимизацию сети включается оптимизация этих базисов. Формально этот слой можно описать одной формулой:

$$F_1(\mathbf{Y}) = \max(0, W_1 * \mathbf{Y} + B_1)$$

где  $W_1$  и  $B_1$  представляют собой фильтры и смещения соответственно, а  $*$  означает операцию свёртки. Здесь,  $W_1$  соответствует  $n_1$  количеству фильтров размером  $(c, f_1, f_1)$ , где  $c$  – это количества каналов входного изображения, а  $f_1$  – пространственный размер фильтра. Отсюда понятно, что  $W_1$  применяет  $n_1$  свёрток к изображению, и каждая свёртка имеет ядро  $(c, f_1, f_1)$ . Выход состоит из  $n_1$  карты признаков. Смещение  $B_1$  – это вектор размерности  $n_1$ , где каждый элемент связан с фильтром. И к результату свёртки применяется функция активации ReLU (Rectified Linear Unit).

## Нелинейное отображение

Первый слой извлёк признаки размерности  $n_1$  для каждого патча. Во второй операции каждый вектор признаков размерности  $n_1$  отображается в вектор признаков размерности  $n_2$ . Это эквивалентно применению  $n_2$  фильтров, которые имеют пространственные размеры  $1 \times 1$ . Эта интерпретация справедлива только для фильтров размерности  $1 \times 1$ . Но это можно легко обобщить на фильтры больших размеров, например,  $3 \times 3$  или  $5 \times 5$ . В этом случае нелинейное отображение находится не на патче из входного изображения, а на  $3 \times 3$  или  $5 \times 5$  патче карты признаков. Операция второго слоя записывается как:

$$F_2(\mathbf{Y}) = \max(0, W_2 * F_1(\mathbf{Y}) + B_2)$$

Здесь  $W_2$  состоит из  $n_2$  фильтров размерности  $(n_1, f_2, f_2)$  и смещения  $B_2$  вектора размерности  $n_2$ . Каждый вектор размерности  $n_2$  является представлением патча высокого разрешения, который будет использован для реконструкции. Также можно добавить

больше свёрточных слоёв для усиления нелинейности, но это может увеличить сложность модели и понадобиться больше времени для обучения.

## Реконструкция

В обычных методах полученные перекрывающимися патчи высокого разрешения часто усредняются для получения окончательного результирующего изображения. Усреднение может быть рассмотрено как предопределенный фильтр на множестве карт признаков, где каждая позиция “плоской” векторной формы патча высокого разрешения. Основываясь на этом можно определить свёрточный слой для получения конечного изображения высокого разрешения:

$$F(\mathbf{Y}) = W_3 * F_2(\mathbf{Y}) + B_3$$

Здесь  $W_3$  состоит из с фильтров размером ( $n_2, f_3, f_3$ ) и смещения  $B_3$  размерности  $c$ . Если представления патчей высокого разрешения в пространстве изображения (то есть можно просто изменить размер каждого представления для формирования патча) можно ожидать, что фильтр ведет себя как усредняющий. Если представления находятся в других пространствах (например, коэффициенты в терминах некоторого базиса) можно ожидать, что  $W_3$  ведет себя как проектирующие отображение на пространство изображения, а затем усреднение. В любом случае  $W_3$  – это набор линейных фильтров.

Интересно получается, что все три операции мотивированными разными представлениями, однако они приводят к одной и той же форме, что и свёрточные слои. Объединими все три операции вместе в одну свёрточную нейронную сеть (как на Рисунке 2). В этой модели все веса фильтров и смещения должны быть оптимизированы. Несмотря на сжатость всей структуры, такая модель SRCNN тщательно разработана с учетом накопленного большого опыта авторов.

## 2.3 Обучение

Обучение полной отображающей функции  $F$  требует оценки параметров сети  $\{W_1, W_2, W_3, B_1, B_2, B_3\}$ . Этого можно достичь с помощью минимизации ошибки между реконструированным изображением и соответствующему реальному изображению высокого разрешения. Используя созданный набор изображений высокого разрешения и соответствующих им изображений низкого разрешения с среднеквадратичной функцией (MSE) ошибки, задачу минимизации можно записать:

$$L(\Theta) = \frac{1}{n} \sum_{i=1}^n \|F(\mathbf{Y}_i; \Theta) - \mathbf{X}_i\|^2$$

где  $n$  – это количество образцов в выборке. Использование MSE способствует высокому значению PSNR. PSNR – это широко используемый показатель для качественной оценки восстановленного изображения, и хотя бы частично связан с качеством восприятия. Несмотря на то, что обучение направлено на увеличение PSNR, также наблюдается улучшение и в других метриках, как SSIM, MSSIM и др.

Ошибка минимизируется с помощью стохастического градиентного спуска со стандартным алгоритмом обратного распространения. Веса фильтров изменяются соответственно формулам:

$$\Delta_{i+1} = 0.9 \cdot \Delta_i - \eta \cdot \frac{\partial L}{\partial W_i^\ell}, \quad W_{i+1}^\ell = W_i^\ell + \Delta_{i+1}$$

На этапе обучения из исходных изображений обучающей выборки вырезался случайный кусок размером ( $f_{\text{sub}}, f_{\text{sub}}, c$ ) изображения с высоким разрешением. Для создания образца низкого разрешения размывается изображение Гауссовым ядром и уменьшается разрешение на порядок, а потом увеличивается на этот же порядок с помощью бикубической интерполяции.

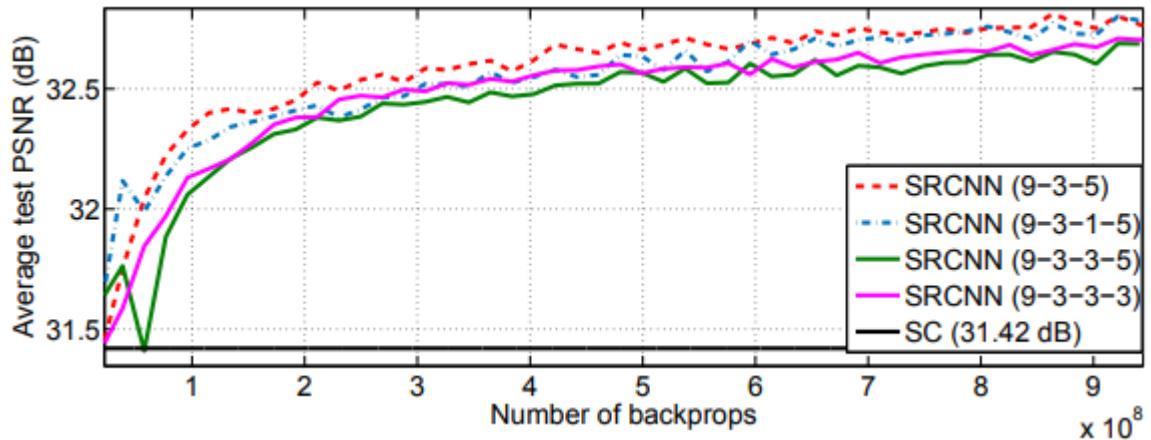
Для избежания краевых эффектов в течение обучения все свёрточные слои не используют отступы (паддинги), и сеть производит меньший выход размером ( $(f_{\text{sub}} - f_1 -$

$f2-f3+3)^2$ , с). Функция потери MSE оценивается только по разнице между центральными пикселями исходного изображения и выходом сети. Хоть обучение происходит на изображениях фиксированного размера, свёрточная сеть может быть применима к изображениям любого размера.

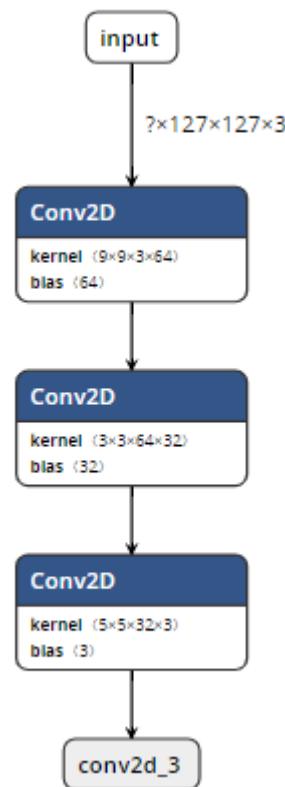
## 2.4 Эксперименты

Для обучения использовались датасеты T91, BSDS200, General100 (<http://vllab.ucmerced.edu/wlai24/LapSRN/>). Из этих датасетов на каждом изображении вырезались куски размером 127x127 – это оригинальные изображения высокого разрешения, потом этот кусок уменьшался до 63x63, а затем увеличили разрешение обратно до 127x127 с помощью бикубической интерполяции. Получили датасет состоящий из пар изображений {X, Y}, где X – изображение, полученное бикубической интерполяцией, а Y – изображение высокого разрешения, которое нужно получить с помощью нейронной сети, то есть получить функцию F, которая будет описывать  $Y = F(X)$ . Из этих датасетов получилось  $\sim 11t$ . пар изображений.

В структуру сети внесены небольшие изменения, на последний слой добавлена функция активации  $tanh$ . А также все изображения предобрабатываются так, чтобы значения каналов лежали в области  $[-1, 1]$ . А размеры фильтров выбраны из исследования авторов исходной статьи (см. Рисунок 3):  $f1=9$ ,  $f2=3$ ,  $f3=5$ ,  $n1=64$ ,  $n2=32$ .



*Рисунок 3. Исследование размеров сети*



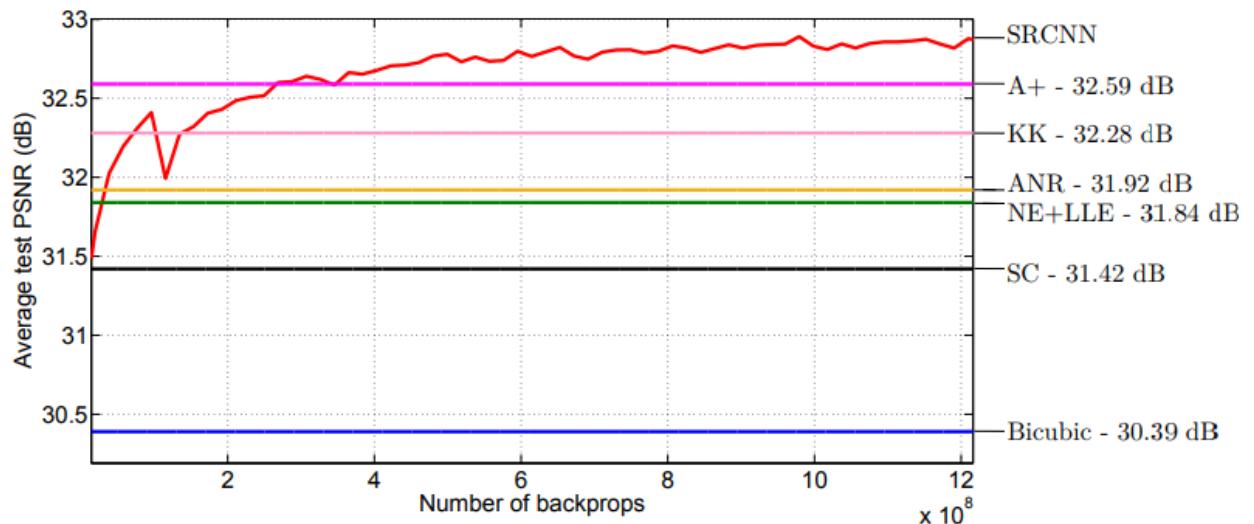
*Рисунок 4. Структура сети*

Также рассмотрим сравнение авторов SRCNN с основными методами суперразрешения:

- SC – Sparse Coding-based method,
- NE+LLE – Neighbor Embedding + Locally Linear Embedding method,

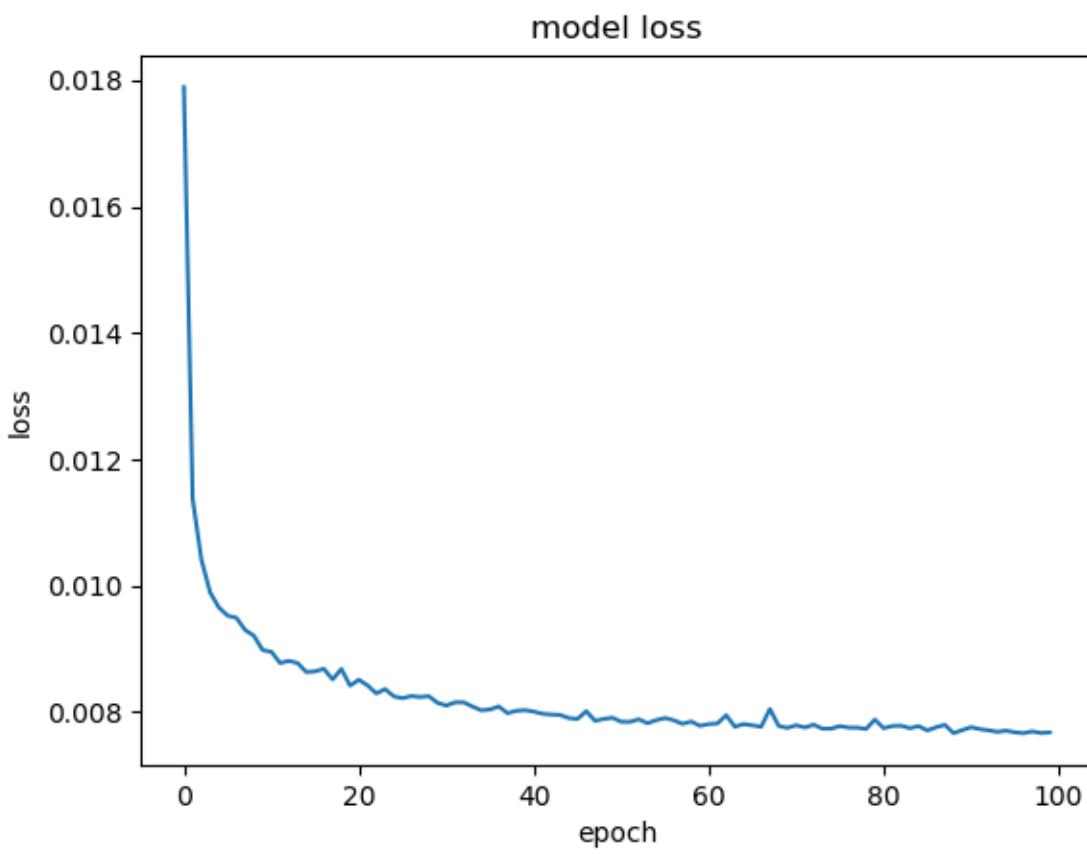
- ANR – Anchored Neighborhood Regression method,
- A+ - Adjusted Anchored Neighborhood Regression method,
- KK – method described in Kim, K.I., Kwon, Y.: Single-image super-resolution using sparse regression and natural image prior.

Сравнение производилось на датасете Set5 (5 изображений). Метрикой для сравнения была использована PSNR. Сравнение отображено на Рисунке 4. Своё преимущество SRCNN показывает, когда сеть достаточно обучилась, уже пройдя  $3 * 10^8$  шагов обучения, сеть превосходит результаты всех основных методов суперразрешения.



*Рисунок 5. Сравнение SRCNN с другими популярными методами SR*

Обучение заняло около одного часа на GTX 1050Ti. Размер обучающей выборки 10338 изображений, а тестовой 1149. В конце обучения получились ошибки на обучающем наборе 0.0077 и на тестовом наборе 0.0076 (Рисунок 5). Эти ошибки достаточно близки, что означает хорошие результаты обучения.



*Рисунок 6. Ошибка на обучающем наборе при обучении*

Рассмотрим также несколько примеров получившихся картинок:

Исходная картинка высокого разрешения	Бикубическая интерполяция	SRCNN
		

**PSNR=20.28 dB**

**PSNR=22.55 dB**

		
	<b>PSNR=30.16 dB</b>	<b>PSNR=31.27 dB</b>
		
	<b>PSNR=24.80 dB</b>	<b>PSNR=25.81 dB</b>
		
	<b>PSNR=20.13 dB</b>	<b>PSNR=21.31 dB</b>

## 2.5 Заключение

Представлен новый подход глубокого обучения для одиночного суперразрешения. Показано, что традиционные методы могут быть преобразованы в глубокие свёрточные нейронные сети. Такой подход, SRCNN, обучается отображению между изображениями низкого и высокого разрешения с небольшой пред- и постобработкой, кроме оптимизации.

Благодаря легкой структуре SRCNN достигает превосходной производительности, по сравнению с другими популярными методами. Можно предположить, что дополнительная производительность может быть получена путём изучения большего количества фильтров и различных стратегий обучения. Кроме того, такая структура, с ее преимуществами простоты и надёжности, может быть применена к другим низкоуровневым проблемам компьютерного зрения, как удаление размытых артефактов или одновременное суперразрешение и шумоподавление. Можно также использовать сеть, чтобы справиться с различными масштабирующими коэффициентами.

Код с архитектурой сети и обучением можно найти в репозитории на GitHub по ссылке <https://github.com/Pol22/SuperResolutionCNN>

## 3. Генеративно-состязательные сети

### 3.1 Введение

Обещание глубинного обучения - открыть богатые, иерархические модели, которые представляют распределение вероятности по типам данных, встречающихся в приложениях искусственного интеллекта таких, как естественные изображения, аудио волны, содержащие голос и символы в естественных языках. До сих пор самые яркие успехи в глубинном обучении имеют дискриминационные модели, которые обычно сопоставляют многомерные данные с меткого класса. Эти поразительные успехи были в основном основаны на алгоритмах обратного распространения ошибки и dropout'a, используя кусочно-линейные блоки, которые имеют особенно выделяющийся градиент. Глубокие генеративные модели имели меньший всплеск из-за сложности аппроксимации многих трудноразрешимых вероятностных исчислений, которые возникают при оценке максимального правдоподобия и связанных стратегий, а также из-за трудности использования преимуществ кусочно-линейных блоков в генеративном контексте. Мы

предлагаем новую процедуру оценки генеративной модели, которая обходит стороной эти сложности.

В предложенном подходе состязательных сетей, генеративная модель противостоит сопернику: дискриминирующая модель, которая обучается определять какой образец из распределения модели, а какой из распределения данных. Генеративную модель можно рассматривать, как аналогию команде фальшивомонетчиков, пытающихся создать поддельные деньги и использовать их без обнаружения, пока дискриминирующая модель аналогичная полиции пытается обнаружить фальшивые деньги. Соревнование в этой игре состоит в том, чтобы обе команды усовершенствовали свои методы до тех пор, пока подделки не будут отличимые от подлинника.

Этот фреймворк может дать определенный алгоритм обучения для многих моделей и оптимизационных алгоритмов. В этой статье, мы исследуем особый случай, когда генеративная модель генерирует образцы путем передачи случайного шума через нейронную сеть, и дискриминирующая модель также нейронная сеть. Мы рассматриваем этот частный случай как состязательную сеть. В этом случае мы можем обучать обе модели, используя только очень успешные алгоритмы обратного распространения ошибки и dropout и образец из генеративной модели, используя прямое распространение. Никаких аппроксимационных выводов или цепей Маркова не нужно.

## 3.2 Главная идея

Структура состязательного моделирования наиболее проста для применения, когда обе модели нейронные сети. Чтобы узнать распределение генератора  $p_g$  по данным  $x$ , мы определяем переменную предыдущего шума на входе  $p_z(z)$ , затем представляем отображение в пространство данных как  $G(z, \theta_g)$ , где  $G$  - это дифференцируемая функция, представленная нейронной сетью с параметрами  $\theta_g$ . Мы также определяем вторую нейронную сеть  $D(x, \theta_g)$ , которая выводит один скаляр.  $D(x)$  представляет

вероятность того, что  $x$ , получен из данных, а не из  $p_g$ . Мы тренируем  $D$  максимизировать вероятность присвоения метки как обучающим примерам и образцам из  $G$ . Мы одновременно тренируем  $G$  минимизировать  $\log(1 - D(G(z)))$ , т.е.  $D$  и  $G$  играют в минимакс игру со значением функции  $V(G, D)$ :

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))] \quad (1)$$

### 3.3 Алгоритм обучения

Генератор  $G$  неявно определяет распределение вероятности  $p_g$  как распределение выборок  $G(z)$ , полученных при  $z \sim p_z$ . Поэтому мы хотели бы, чтобы Алгоритм сходился к хорошей оценке  $p_{\text{data}}$ , если ему была предоставлена достаточная емкость и время обучения. Алгоритм из этой главы выполняет параметрическую настройку, например, мы представляем модель с бесконечной емкостью, изучая сходимость в пространстве функций плотности вероятности.

Эта минимакс игра имеет глобальный оптимум для  $p_g = p_{\text{data}}$ , а также алгоритм оптимизирует выражение (1), получая таким образом желаемый результат.

---

**Алгоритм** Обучение стохастическим градиентным спуском на мини пакетах генеративно-состязательных сетей. Число шагов, применяемых к дискrimинатору,  $k$ , это гиперпараметр.

---

```

for number of training iterations do
  for  $k$  steps do
    • Sample minibatch of  $m$  noise samples  $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  from noise prior  $p_g(\mathbf{z})$ .
    • Sample minibatch of  $m$  examples  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  from data generating distribution  $p_{\text{data}}(\mathbf{x})$ .
    • Update the discriminator by ascending its stochastic gradient:
      
$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[ \log D(\mathbf{x}^{(i)}) + \log (1 - D(G(\mathbf{z}^{(i)}))) \right].$$

  end for
  • Sample minibatch of  $m$  noise samples  $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  from noise prior  $p_g(\mathbf{z})$ .
  • Update the generator by descending its stochastic gradient:
    
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log (1 - D(G(\mathbf{z}^{(i)}))).$$

end for

```

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

## 3.4 Эксперименты

В работе Goodfellow et al. обучили состязательные сети на ряде наборов данных, включая MNIST, база данных лиц Торонто и CIFAR-10. В сетях генератора использовалась смесь линейных функций активации и сигмоид, пока сеть дискриминатора использовала maxout активацию. Дропаут был применен при обучении сети дискриминаторов. Пока наши теоретические основы позволяют использовать дропаут и другие шумы на промежуточных слоях генератора, мы использовали шум в качестве входа для нижнего слоя сети генератора.

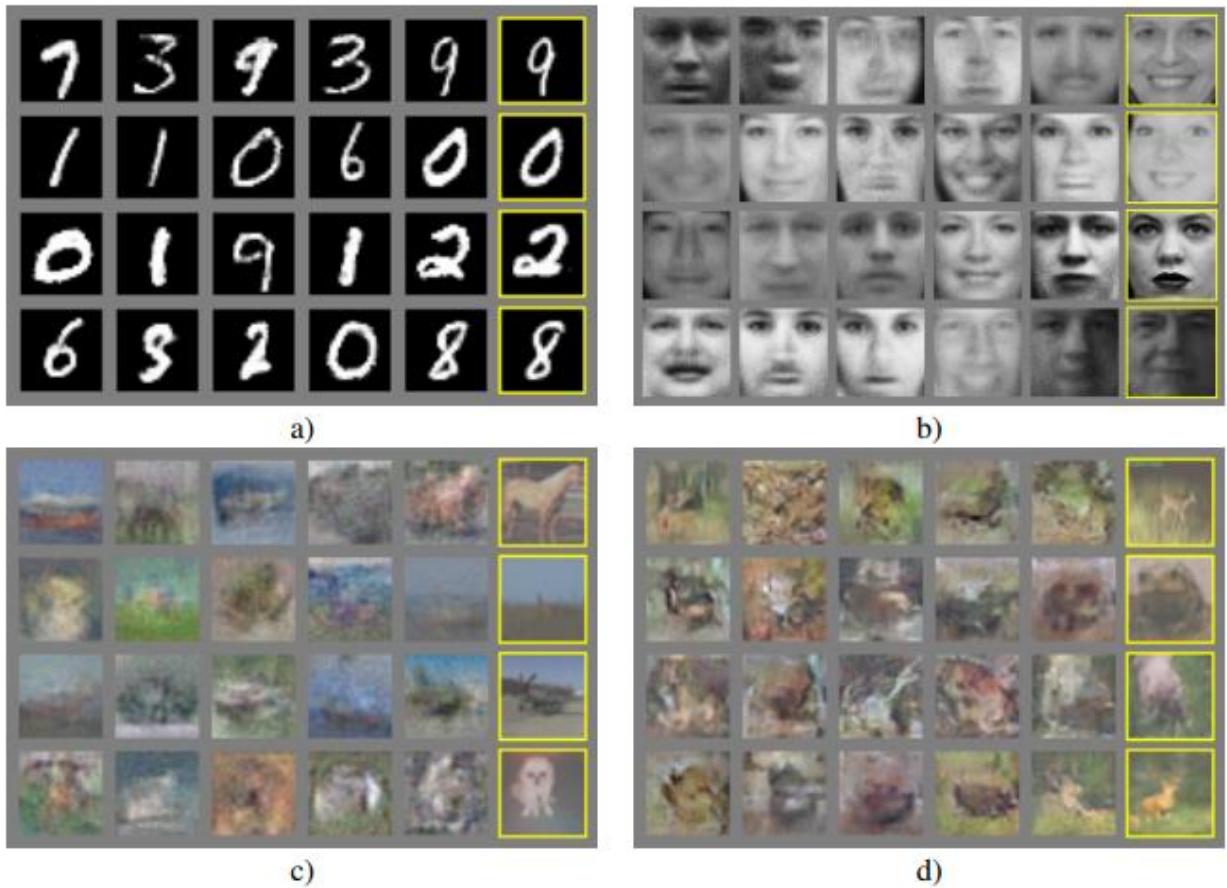
В работе оценивали вероятность тестового набора данных на  $p_g$  путем подгонки окна Gaussian Parzen к образцам, сгенерированным с помощью  $G$  и сообщая логарифмическую вероятность в рамках этого распределения. Параметр  $\sigma$  гауссианов был получен путем перекрестной проверки на множестве валидаций. Эта процедура была введена в Breulex и используется для различных генеративных моделей, для которых точная вероятность не вычисляется. Результаты приведены в Таблице 1. Этот метод

оценки вероятности имеет большую вариацию и плохо работает в многомерных пространствах, но это лучший метод доступный нашим знаниям. Достижения в генеративных моделях, которые могут делать образцы, но не оценивать вероятность напрямую для дальнейшего использования, как оценивание таких моделей.

Model	MNIST	TFD
DBN [3]	$138 \pm 2$	$1909 \pm 66$
Stacked CAE [3]	$121 \pm 1.6$	$2110 \pm 50$
Deep GSN [6]	$214 \pm 1.1$	$1890 \pm 29$
Adversarial nets	$225 \pm 2$	$2057 \pm 26$

*Таблица 1: Оценки логарифмического правдоподобия на основе окна Parzen. Числа в MNIST означают среднюю логарифмическую вероятность образцов тестовой выборки со стандартной ошибкой среднего значения, вычисленного на примерах. В TFD мы вычислили стандартную ошибку по выборкам наборов данных с другим  $\sigma$ , выбранным по проверочной выборке каждого сета. В TFD  $\sigma$  была перекрестно проверена на каждой выборке и была посчитана логарифмическая вероятность на каждую выборку. Кроме MNIST сравнивали другие модели вещественной (а не двоичной) версии набора данных.*

На Рисунке 1 показаны образцы, взятые с сети генератора после обучения. Пока не утверждаем, что эти образцы лучше, чем образцы, сгенерированные существующими методами, мы верим, что эти образцы как минимум конкурентоспособны с лучшими генеративными моделями в литературе и подчеркивают потенциал состязательной архитектуры.



*Рисунок 1: Визуализация образцов из модели. В самом правом столбце показан ближайший пример обучения из соседних образцов, чтобы показать, что модель не запомнила сет обучения. Образцы — это честные случайные розыгрыши, а не хорошо выбранные. В отличии от большинства визуализаций глубинных генеративных моделей, эти изображения показывают фактические образцы из модели распределений, а не условные образцы скрытых объектов. Более того, эти образцы некоррелированные, так как процесс отбора образцов не зависит от цепного смещивания Маркова. a) MNIST b) TFD c) CIFAR-10 (полносвязная модель) d) CIFAR-10 (сверточный дискриминатор и «разверточный» генератор)*

### 3.5 Преимущества и недостатки

Эта новая структура имеет преимущества и недостатки по сравнению с предыдущими структурами моделирования. Недостатки в первую очередь состоят в том, что нет явного представления  $p_g(x)$  и, что D должен хорошо синхронизироваться с G в процессе обучения (в частности, G не должен быть обучен слишком много без обновления D, чтобы избежать «сценария Helvetica», в котором G сбрасывает слишком много

значений  $z$  одновременно со значением  $x$ , чтобы иметь достаточное пространство для модели  $p_{data}$ ), так как отрицательные цепочки Больцмана должны быть обновлены между этапами обучения. Преимущества заключаются в том, что цепи Маркова не нужны, только для обратного распространения используется полученные градиенты, во время обучения не требуются выводы, и в модель могут быть включены разнообразные функции. В Таблице 2 приведено сравнение генеративно-состязательных сетей с другими генеративными подходами к моделированию.

Вышеупомянутые преимущества в первую очередь вычислительные. Состязательные модели также могут получить некоторое статистическое преимущество из сети генератора, которая не обновляется непосредственно с примерами данных, а только с градиентами, протекающими через дискриминатор. Это означает, что компоненты входа не копируются непосредственно в параметры генератора. Другим преимуществом состязательных сетей является то, что они могут представить очень резкие, даже дегенеративные распределения тогда, как методы, основанные на цепях Маркова, требуют, чтобы распределение было несколько размытым, чтобы цепи могли смешиваться между режимами.

## 4. Практическое использование генеративно-состязательных сетей

### 4.1 Введение

Генеративное моделирование предполагает аппроксимацию невычислимых апостериорных распределений. Из-за этого большинство эффективных методов, разработанных для обучения дискриминативных моделей, не работают с генеративными моделями. Существующие в прошлом методы для решения этой задачи вычислительно трудны и, в основном, основаны на использовании [методов](#) Монте Карло с цепями

Маркова, которые плохо масштабируются. Поэтому для обучения генеративных моделей нужен был метод, основанный на таких масштабируемых техниках, как стохастический градиентный спуск и метод обратного распространения ошибки. Один из таких методов — Генеративно-состязательные сети (GAN = Generative Adversarial Network). Впервые GANы были предложены в 2014 году. На высоком уровне эта модель может быть описана, как две подмодели, которые соревнуются друг с другом, и одна из этих моделей (генератор), пытается научиться в некотором смысле обманывать вторую (дискриминатор). Для этого генератор генерирует случайные объекты, а дискриминатор пытается отличить эти сгенерированные объекты от настоящих объектов из тренировочной выборки. В процессе обучения генератор генерирует все более похожие на выборку объекты и дискриминатору становится все сложнее отличить их от настоящих. Таким образом, генератор превращается в генеративную модель, которая генерирует объекты из некого сложного распределения, например, из распределения фотографий человеческих лиц.

## 4.2 Модель

Для начала введем необходимую терминологию. Через  $X$  мы будем обозначать некоторое пространство объектов. Например, картинки  $64 * 64 * 3$  пикселя. На некотором вероятностном пространстве  $\Omega$  задана векторная случайная величина  $x : \Omega \rightarrow X$  с распределением вероятностей, имеющим плотность  $p(x)$  такую, что подмножество пространства  $X$ , на котором  $p(x)$  принимает ненулевые значения — это, например, фотографии человеческих лиц. Нам дана случайная независимая одинаково распределенная выборка фотографий лиц для величины  $\{x_i, i \in [1, N], x_i \sim p(x)\}$ . Дополнительно определим вспомогательное пространство  $Z = R^n$  и случайную величину  $z : \Phi \rightarrow Z$  с распределением вероятностей, имеющим плотность  $q(z)$ .  $D : X \rightarrow (0, 1)$  — функция-дискриминатор. Эта функция принимает на вход объект  $x \in X$  (в нашем примере — картинку соответствующего размера) и возвращает вероятность того, что входная картинка является фотографией

человеческого лица.  $G : Z \rightarrow X$  — функция-генератор. Она принимает значение  $z \in Z$  и выдает объект пространства  $X$ , то есть, в нашем случае, картинку.

Предположим, что у нас уже есть идеальный дискриминатор  $D$ . Для любого примера  $x$  он выдает истинную вероятность принадлежности этого примера заданному подмножеству  $X$ , из которого получена выборка  $\{x_i\}$ . Переформулируем задачу обмана дискриминатора на вероятностном языке мы получаем, что необходимо максимизировать вероятность, выдаваемую идеальным дискриминатором на сгенерированных примерах.

Таким образом оптимальный генератор находится как  $G^* = \arg \max_G E_{z \sim q(x)} D_k(G(z))$ .

Так как  $\log(x)$  — монотонно возрастающая функция и не меняет положения экстремумов аргумента, эту формулу переписать в виде  $G^* = \arg \max_G E_{z \sim q(x)} \log D_k(G(z))$ , что будет удобно в дальнейшем.

В реальности обычно идеального дискриминатора нет и его надо найти. Так как задача дискриминатора — предоставлять сигнал для обучения генератора, вместо идеального дискриминатора достаточно взять дискриминатор, идеально отделяющий настоящие примеры от сгенерированных текущим генератором, т.е. идеальный только на подмножестве  $X$  из которого генерируются примеры текущим генератором. Эту задачу можно переформулировать, как поиск такой функции  $D$ , которая максимизирует вероятность правильной классификации примеров как настоящих или сгенерированных. Это называется задачей бинарной классификации и в данном случае мы имеем бесконечную обучающую выборку: конечное число настоящих примеров и потенциально бесконечное число сгенерированных примеров. У каждого примера есть метка: настоящий он или сгенерированный. В оригинальной статье было описано решение задачи классификации с помощью метода максимального правдоподобия:

$$\min_G \max_D L(D, G) = E_{x \sim p(x)} \log D(x) + E_{z \sim q(z)} \log (1 - D(G(z)))$$

где наша выборка  $S = \{(x, 1), x \sim p(x)\} \cup \{(G(z), 0), z \sim q(z)\}$ .

Тогда мы получаем итерационный процесс:

1. Устанавливается произвольный начальный генератор  $G_0(z)$ .

2. Начинается  $k$ -я итерация,  $k = 1 \dots K$ .

3. Ищем оптимальный дискриминатор для текущего генератора:

$$D_k = \arg \max_D E_{x_i \sim p(x)} \log D(x_i) + E_{z_i \sim q(z)} \log (1 - D(G_{k-1}(z_i)))$$

4. Улучшаем генератор, используя оптимальный дискриминатор:

$$G_k = \arg \max_G E_{z \sim q(x)} \log D_k(G(z))$$

Важно находиться в окрестности текущего генератора. Если отойти далеко от него, то дискриминатор перестанет быть оптимальным и алгоритм разойдется.

Задача обучения будет законченной, если  $D_k(x) = 1/2$  для любого  $x$ . Иначе, переходим к пункту (2).

Обе функции  $D, G$  могут быть представлены в виде нейросетей:  $D(x) = D(x, \theta_1), G(z) = G(z, \theta_2)$ , после чего задача поиска оптимальных функций сводится к задаче оптимизации по параметрам и ее можно решать с помощью традиционных методов: стохастического градиентного спуска и метода обратного распространения ошибки. Дополнительно, так как нейросеть — это универсальный аппроксиматор функций,  $G(z, \theta_2)$  может приблизить произвольное распределение вероятностей, что снимает вопрос выбора распределения  $q(z)$ . Это может быть любое непрерывное распределение в некоторых разумных рамках. Например, равномерное или нормальное. Корректность этого алгоритма и сходимость  $G(z)$  к  $p(x)$  при достаточно общих предположениях доказана в оригинальной статье.

## 4.3 Задача нахождения параметров нормального распределения

Давайте посмотрим, как это работает. Допустим,  $X = R$ , т.е. решаем одномерную задачу.  $p(x) = N(\mu, \sigma)$ ,  $q(z) = N(0, 1)$ . Давайте использовать линейный генератор  $G(z, \theta) = az + b$ , где  $\theta = \{a, b\}$ . Дискриминатор будет полносвязной трехслойной нейронной сетью с бинарным классификатором на конце. Решением этой задачи является  $G(z, \mu, \sigma) = \mu z + \sigma$ , то есть,  $a = \mu, b = \sigma$ . Попробуем теперь запрограммировать численное решение этой задачи на языке Python с помощью библиотеки Tensorflow.

Первое, что нужно задать, это входную выборку:  $p(x) = N(\mu, \sigma)$ . Так как обучение идет на минибатчах, мы будем за раз генерировать вектор чисел реальных. Дополнительно, выборка параметризуется средним и стандартным отклонением.

Теперь определим генератор в виде нейронной сети. У него на входе  $q(z) = N(0, 1)$ :

Теперь определим дискриминатор в виде полносвязной нейронной сети, например, с 3 скрытыми слоями. Создадим также вектор сгенерированных примеров и пронгим все примеры через дискриминатор.

Функция потерь на реальных примерах – это кросс-энтропия между единицей (положительным ответом дискриминатора на реальных примерах) и оценками дискриминатора.

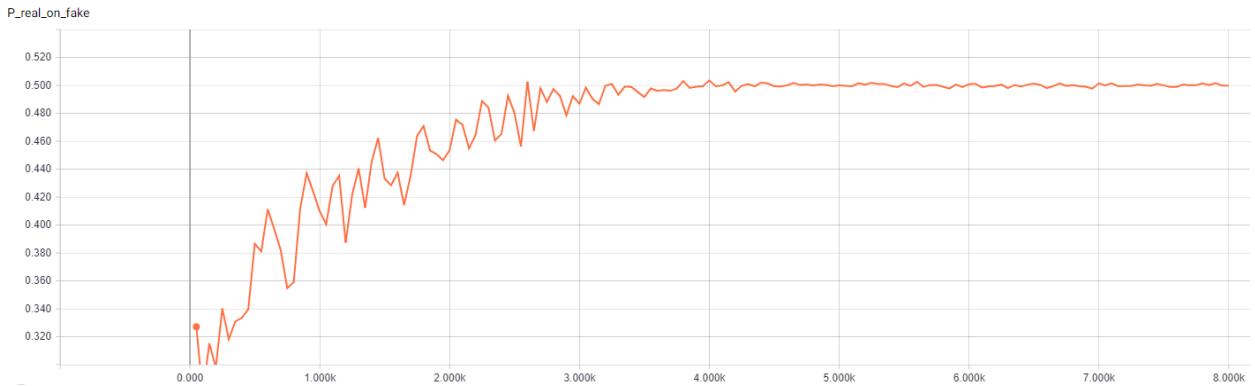
Функция потерь на сгенерированных примерах – это кросс-энтропия между нулем (отрицательным ответом дискриминатора на поддельных примерах) и оценками дискриминатора.

Тогда функция потерь дискриминатора будет равна сумме потерь на реальных и поддельных примерах. А функция потерь генератора – это кросс-энтропия между

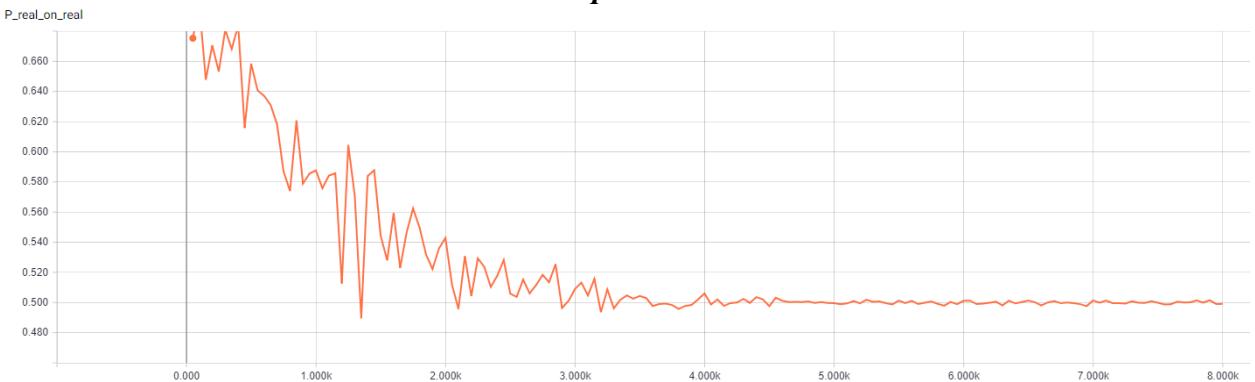
единицей (положительным ответом дискриминатора на поддельных примерах) и оценками дискриминатора:

Обучение модели сводится к одновременному обучению дискриминатора и генератора в цикле до сходимости, только при обучении дискриминатора могут быть задействованы параметры генератора, их следует убрать из переменных обучения.

Посмотрим теперь графики обучения сетей в зависимости от шага обучения:

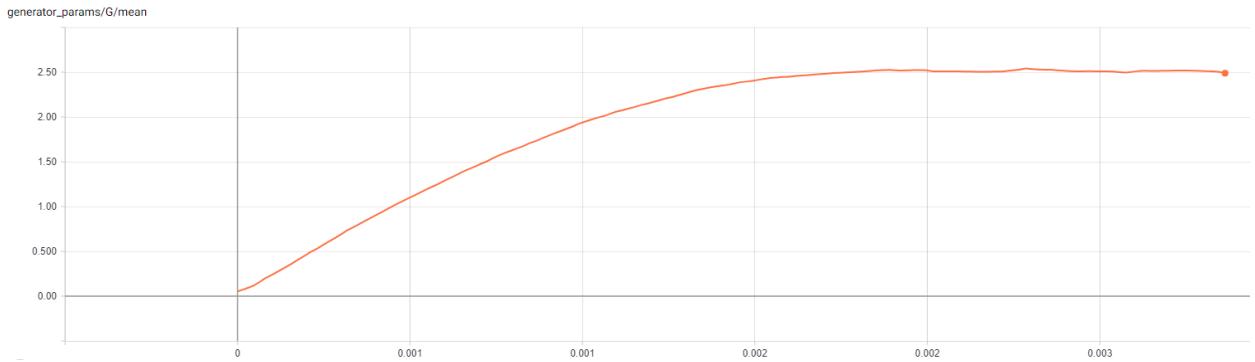


**Рисунок 1. Вероятность классификации дискриминатором сгенерированного примера как реального**

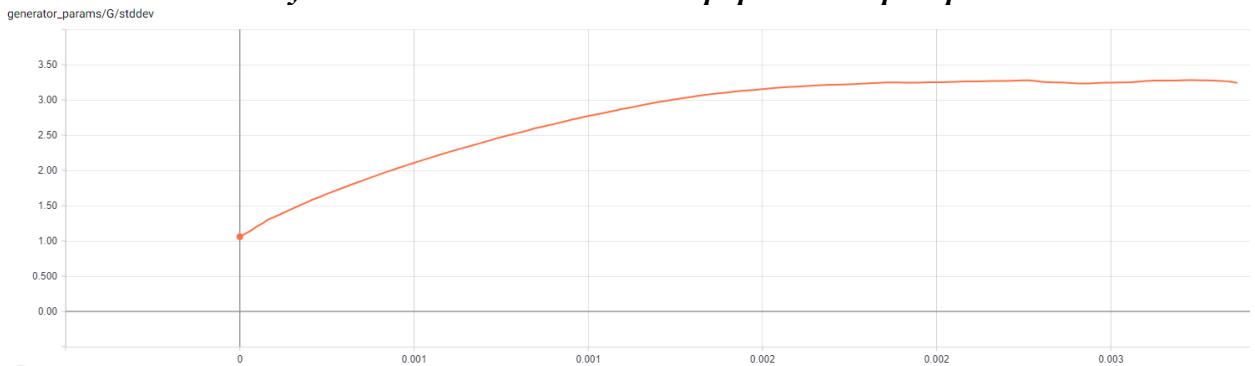


**Рисунок 2. Вероятность классификации дискриминатором реального примера как реального**

Модель достаточно быстро сходится к тому, что дискриминатор выдает  $\frac{1}{2}$  при любых данных на входе.



*Рисунок 3. Мат. ожидание сгенерированных распределений*



*Рисунок 4. Дисперсия сгенерированных распределений*

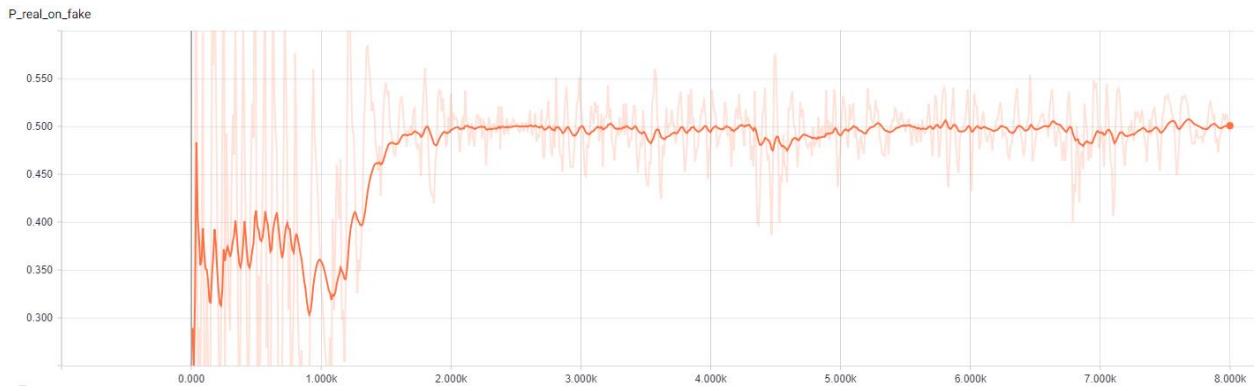
Из-за простоты задачи для генератора из графиков видно, что математическое ожидание и дисперсия быстро сходятся к значениям из распределения данных.

#### **4.4 Задача приближения смеси нормальных распределений**

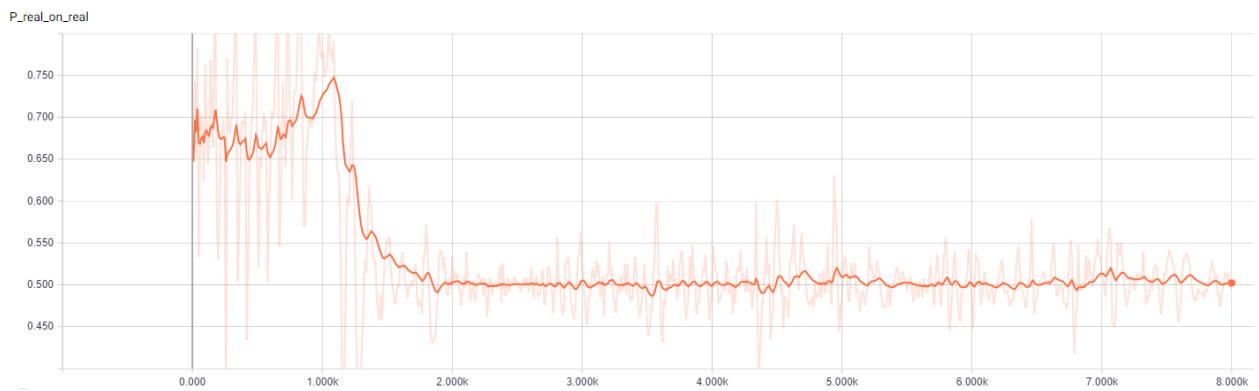
Заменим  $p(x) = N(\mu, \sigma)$  на  $p(x) = Mixture(N(\mu_1, \sigma_1), N(\mu_2, \sigma_2))$ .

Конечно приближение таких данных нормальным распределением с обучаемыми параметрами не даст хорошего результата. Поэтому следует улучшить структуру генератора, например, сделаем ее полносвязной нейронной сетью:

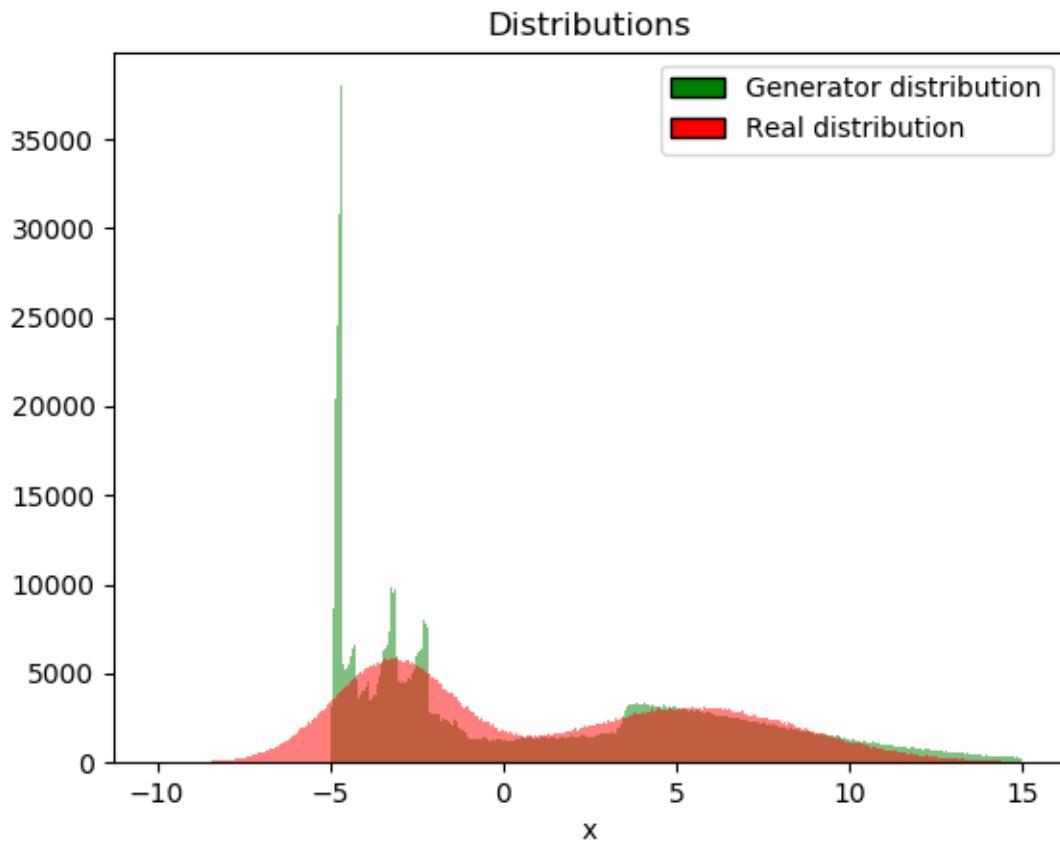
Посмотрим теперь графики обучения сетей в зависимости от шага обучения:



**Рисунок 5. Вероятность классификации дискриминатором сгенерированного примера как реального**



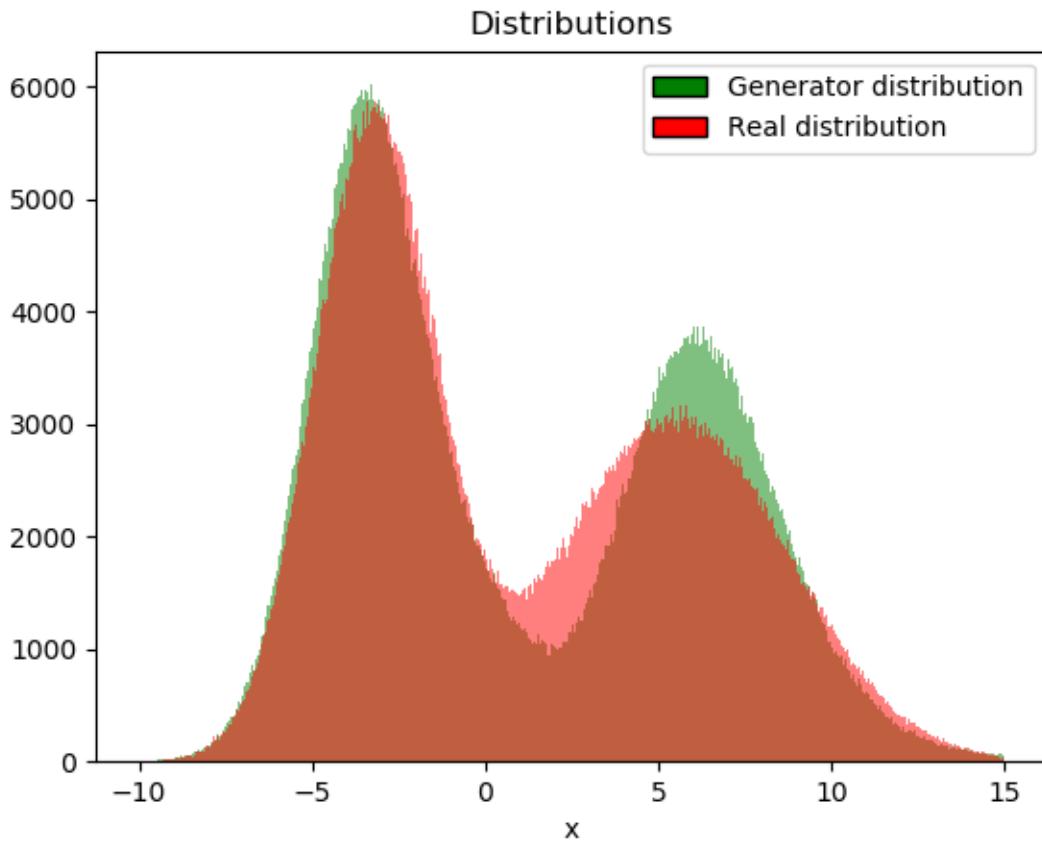
**Рисунок 6. Вероятность классификации дискриминатором реального примера как реального**



*Рисунок 7. Распределения генератора и реальных данных*

Видно, что распределение генератора хоть и не совпадает с распределением данных, но достаточно сильно похоже на него. Таким образом генератор на основе нейронных сетей способен выучить мультимодальное распределение данных.

Попробуем улучшить нашу модель, добавив dropout-регуляризацию между скрытыми слоями нейронной сети генератора. Получили вот такое распределение:



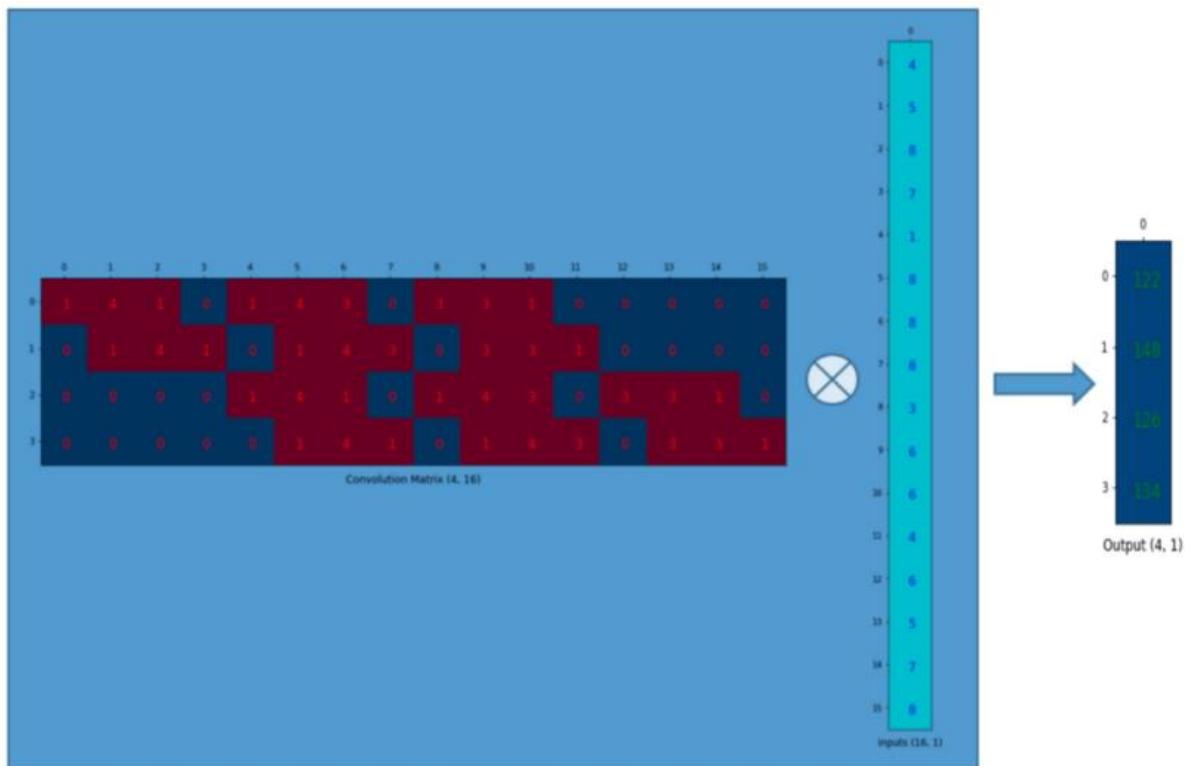
*Рисунок 8. Распределения генератора и реальных данных*

Из распределений видно, что при использовании достаточно сложного генератора с множеством параметров он способен приближать многомодальное распределение. Дискриминаторы получаются более шумные, чем в первом примере, но к концу обучения они также представляют прямую  $D(x) = 1/2$ .

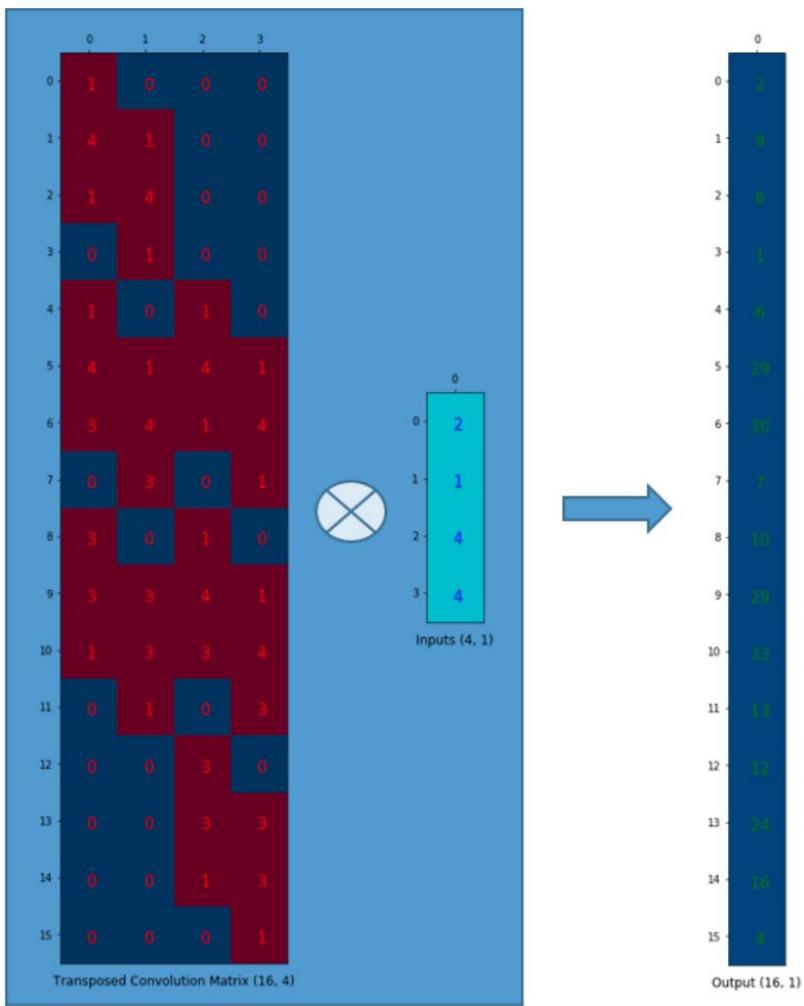
## **4.5 Задача приближения распределения изображений**

Рассмотрим теперь пример, в котором будем приближать генеративной сетью распределение набора бинарных изображений. Для работы с изображениями структуру нейронных сетей необходимо изменить на сверточную. Дискриминатор, например, можно сделать из двух сверточных слоев и одного полносвязного. С генератором немного

сложнее, так как нам надо создавать изображение из вектора случайных чисел. Для этого используются слои транспонированной свертки (transposed convolutional), которые работают обратно обычной свертке – возвращает матрицу значений по одному известному числу. Она имеет такое название, так как в преобразовании используется транспонированная матрица свертки (см. Рисунок 9-10).



*Рисунок 9. Обычная свёртка*



*Рисунок 10. Транспонированная свёртка*

Рассмотрим задачу, что нам нужно генерировать изображения синусоид с различными фазами. Для обучения сетей будем использовать бесконечный (в смысле что можно генерировать сколько угодно изображений) датасет, состоящий из бинарных изображений синусоид размером 50x50 пикселей, у которых фаза распределена равномерно  $R(0, 360)$  (см. Рисунок 11). Генератор будет сверточной нейронной сетью с одним полно связным слоем и двумя слоями транспонированной свёртки, а дискриминатор будет обычной сетью с двумя слоями свёртки и одним полно связным слоем. Также добавим регуляризацию весов каждой нейронной сети, то есть введем штраф в функцию потери за норму всех обучаемых переменных.



*Рисунок 11. Десять изображений из датасета*

Обучение будет происходить на минибатчах размера 32 изображения в каждой итерации. На вход генератора будет подаваться равномерный шум из распределения  $R(-1, 1)$ . Рассмотрим процесс обучения:

Шаг обучения	Пример генерации (10 изображений генератора)
100	
1000	
2000	
5000	

Результаты получаются немного шумными, но это легко исправить фильтрацией или использованием более сложных моделей нейронных сетей генератора. Таким образом получается с помощью генеративно-состязательных сетей можно генерировать изображения любого качества и любой структуры путем обучения сетей на датасетах из представителей желанного класса.

## **4.6 Итоги**

Генеративно-состязательные сети – это модель приближения произвольного распределения только с помощью семплирования этого распределения. Мы рассмотрели в деталях, как работает модель на тривиальном примере поиска параметров нормального распределения и на более сложном примере приближения распределения изображений. Обе задачи достаточно хорошо решаются с большой точностью, для чего потребовалась только более сложная модель генератора.

Код с архитектурой сети и обучением можно найти в репозитории на GitHub по ссылке [https://github.com/Pol22/GAN\\_testing](https://github.com/Pol22/GAN_testing)

# **5. Супер-разрешения с помощью свёрточных генеративно-состязательных сетей**

## **5.1 Введение**

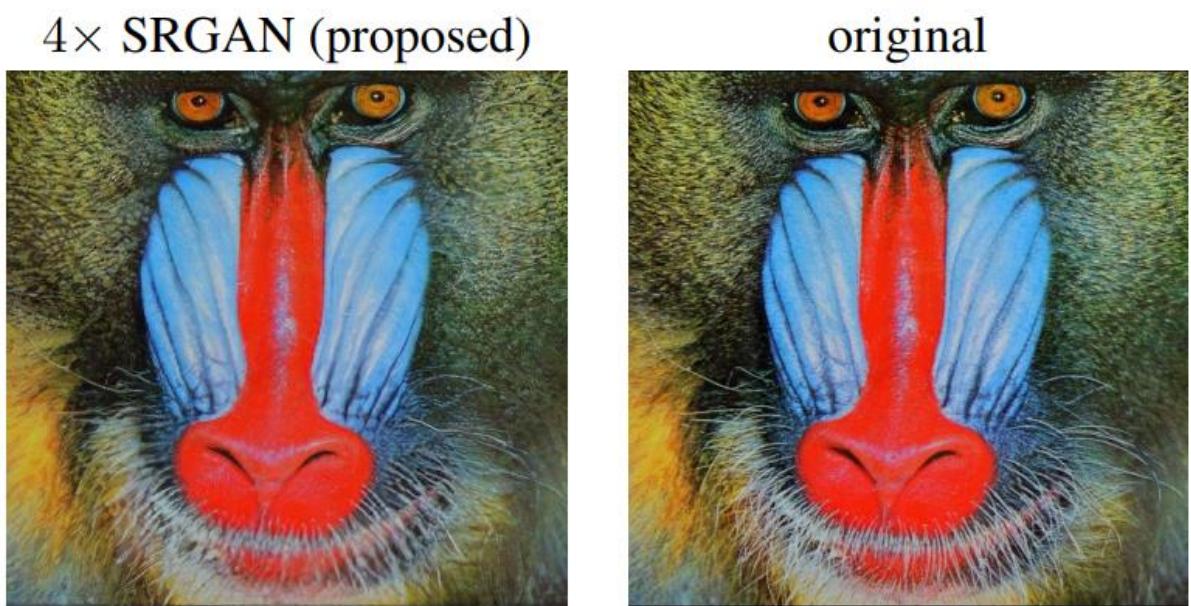
Очень сложная задача получения изображения с высоким разрешением (HR) из его дубликата с низким разрешением (LR), это называется суперразрешением (SR). SR получил значительное внимание со стороны сообщества исследователей компьютерного зрения и имеет широкий спектр применений.

Неправильно спозиционированная природа недоопределенной проблемы SR особенно выражена для больших коэффициентов масштабирования, для которых детали текстуры в реконструированных SR изображениях обычно отсутствуют. Цель оптимизации SR алгоритмов с учителем обычно сводится к минимизации среднеквадратичной ошибки между восстановленным HR изображением и истинным HR изображением. Это удобно, так как минимизация MSE также максимизирует пиковое отношение сигнал-шум (PSNR), которое является стандартной мерой для оценки и сравнения алгоритмов SR. Однако способность MSE (и PSNR) к улавливанию воспринимаемых различий, как высокодетализированные текстуры, очень ограничена, так как они определены на пиксельной разнице. Это показано на Рисунке 1, где самый высокий PSNR не обязательно отображает лучший результат SR по восприятию. Разница восприятия между SR изображением и оригинальным показывает, что восстановленное изображение не является фотoreалистичным, как определено в работе Ferwerda ([http://cin.ufpe.br/~in1123/material/vor\\_hvei03\\_v20.pdf](http://cin.ufpe.br/~in1123/material/vor_hvei03_v20.pdf)). Фотoreалистичное изображение производит тот же визуальный эффект как и реальная сцена).



*Рисунок 7. Слева направо: бикубическая интерполяция, глубокая остаточная сеть, оптимизированная с МНК, глубокая остаточная генеративно-состязательная сеть, оптимизированная для потери более чувствительной к человеческому восприятию, оригинальное HR изображение. Соответствующие PSNR и SSIM показаны в скобках. [4x масштабирование]*

Рассмотрим генеративно состязательную сеть суперразрешения (SRGAN), в которой используем глубокую свёрточную сеть с residual связями (ResNet), с пропущенными соединениями и с целью оптимизации отличающейся от MSE. В отличие от предыдущих работ, определяем новую потерю восприятия, использующую карты особенностей высокого уровня сети VGG в сочетании с дискриминатором, который поощряет результаты, которые трудно отличить от эталонных изображений HR. Пример фотореалистичного изображения, которое было получено масштабированием в 4 раза показано на Рисунке 2.



*Рисунок 28. Суперразрешение изображения (слева) почти неотличимо от оригинального (справа). [4x масштабирование]*

## 5.2 Метод

Сконцентрируемся на суперразрешении одного изображения (SISR = Single Image Super Resolution) и не будем затрагивать темы, покрывающие HR изображения по нескольким изображениям. В SISR целью является оценка высокого разрешения, суперразрешимое изображение  $I^{SR}$  из изображения низкого разрешения  $I^{LR}$ . Здесь  $I^{LR}$  дубликат с низким разрешением версии  $I^{HR}$ . Изображения высокого разрешения

доступны только во время обучения. В течение обучения,  $I^{LR}$  получаются применением фильтра Гаусса перед последующим применением операции понижения дискретизации с коэффициентом  $r$ . Для изображения с  $C$  цветовыми каналами, мы опишем  $I^{LR}$  как вещественный тензор размером  $W * H * C$  и  $I^{HR}, I^{SR}$  как  $rW * rH * C$  соответственно.

Наша конечная цель - это обучить генерирующую функцию  $G$ , которая создает по полученному LR входному изображению его соответствующий HR аналог. Для достижения этого, мы обучаем генеративную сеть как прямонаправленную свёрточную нейронную сети  $G_{\theta_G}$ , зависящую от параметров  $\theta_G$ . Здесь  $\theta_G = \{W_{1:L}, b_{1:L}\}$  обозначает весовые коэффициенты и смещения сети L-го уровня глубокой сети и получаются они путём оптимизации специальной для SR функции потерь  $l^{SR}$ . С тренировочными изображениями  $I_n^{HR}, n = 1, \dots, N$  с соответствующими  $I_n^{LR}, n = 1, \dots, N$ , мы решаем задачу оптимизации:

$$\hat{\theta}_G = \arg \min_{\theta_G} \frac{1}{N} \sum_{n=1}^N l^{SR}(G_{\theta_G}(I_n^{LR}), I_n^{HR}) \quad (1)$$

В этой работе мы специально спроектируем функцию потери восприятия  $l^{SR}$  как взвешенную комбинацию из нескольких компонент, которые моделируют различные желаемые характеристики восстановленного SR изображения.

## Архитектура состязательной сети

Следуя определению Goodfellow мы также определим сеть дискриминатор  $D_{\theta_D}$ , которую мы оптимизируем поочередно наряду с  $G_{\theta_G}$  для решения состязательной мин-макс проблемы:

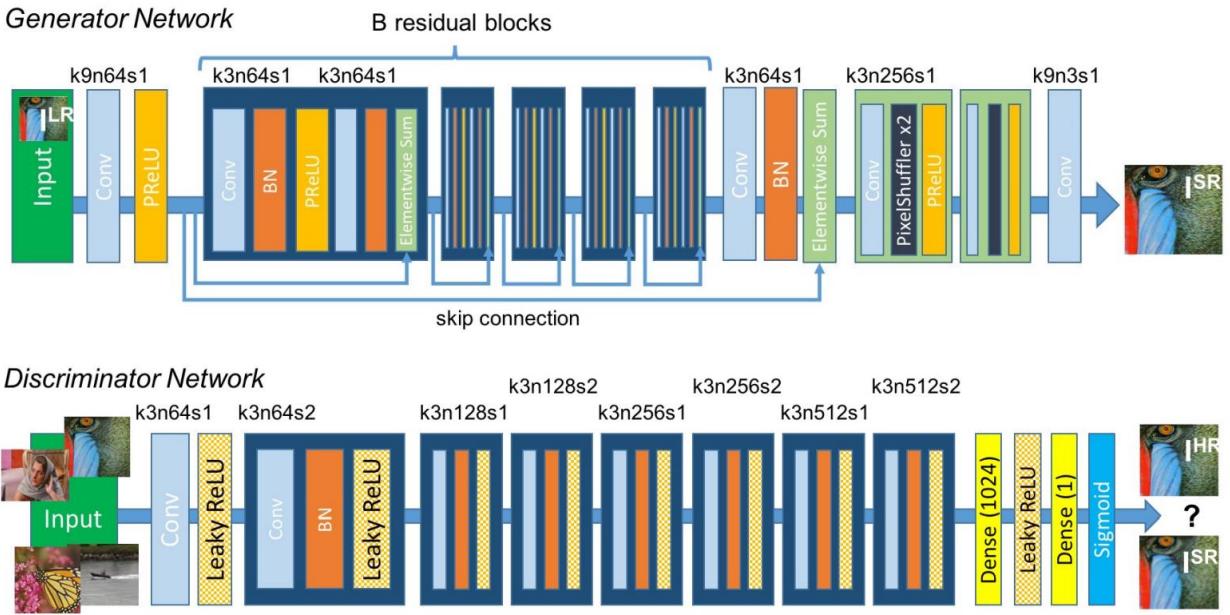
$$\min_{\theta_G} \max_{\theta_D} \mathbb{E}_{I^{HR} \sim p_{\text{train}}(I^{HR})} [\log D_{\theta_D}(I^{HR})] + \mathbb{E}_{I^{LR} \sim p_G(I^{LR})} [\log(1 - D_{\theta_D}(G_{\theta_G}(I^{LR})))] \quad (2)$$

Основная идея, заключенная в этой формуле - это позволить тренироваться генеративной модели  $G$  с основной целью обмануть дискриминатор  $D$ , который тренируется распознавать суперразрешенные изображения от реальных изображений. С этим подходом наш генератор может научиться создавать решения, которые очень похожи на реальные изображения и поэтому их сложно классифицировать дискриминатору  $D$ . Это поощряет решения, которые лучше воспринимаются визуально (перцептивно), находящиеся в подпространстве, многообразии естественных изображений. Это контрастирует с SR решениями, прибегающими к минимизации измерений попиксельной ошибки, как MSE.

В центре нашей очень глубокой генеративной сети  $G$ , которая отображена на Рисунке 3, в которой В количество residual блоков с идентичным строением. В особенности, мы используем два свёрточных слоя с маленьким ядром 3x3 и с 64-мя картами особенностей, отправляющиеся затем в batch-normalization слои и ParametricReLU, как функцию активации. Мы увеличиваем разрешение входного изображения с помощью двух обученных субпиксельных свёрточных слоёв, предложенных Shi et al. (<https://arxiv.org/pdf/1609.05158.pdf>)

Чтобы определить настоящие HR изображения от сгенерированных SR образцов, мы обучаем сеть дискриминатор. Архитектура показана на Рисунке 3. Мы следуем архитектурным рекомендациям резюмированым Radford et al. (<https://arxiv.org/pdf/1511.06434.pdf>), и используем LeakyReLU активацию ( $\alpha = 0.2$ ), и избегаем максимального пулинга во всей сети. Сеть дискриминатор обучена решать задачу максимизации в Выражении 2. Она содержит 8 свёрточных слоёв с увеличивающимся размером ядер от 3x3 с фактором 2, от 64 до 512 фильтров в ядре, как в сети VGG. В свёртке используются strides для уменьшения разрешения изображения,

каждый раз количество особенностей удваивается. В результате 512 карт особенностей проходят через два полно связанных слоя и в конце сигмоидная функция активации для получения вероятностей классификации.



*Рисунок 3. Архитектура Генерирующей и Дискриминирующей сети с соответствующими размерами ядер ( $k$ ), количествами карт особенностей ( $n$ ) и шагами ( $s$ ), которые показаны для каждого свёрточного слоя.*

## Функция потери восприятия

Предназначение нашей функции потери восприятия  $l^{SR}$  имеет решающее значение для производительности нашей генерирующей сети. Пока  $l^{SR}$  обычно смоделировано на основе MSE, мы улучшаем ее с помощью предложений из Johnson et al. (<https://arxiv.org/pdf/1603.08155.pdf>) и Bruna et al. (<https://arxiv.org/pdf/1511.05666.pdf>) и конструируем функцию потерь, которая оценивает решение с точки зрения воспринимательных характеристик. Мы сформулируем потери восприятия, как взвешенная сумма потери содержания  $l_X^{SR}$  и потери состязательной компоненты  $l_{Gen}^{SR}$  как:

$$l^{SR} = \underbrace{l_X^{SR}}_{\text{content loss}} + \underbrace{10^{-3} l_{Gen}^{SR}}_{\text{adversarial loss}} \quad (3)$$

perceptual loss (for VGG based content losses)

Далее мы опишем возможные реализации для потери содержания и состязательной потери.

## Потеря содержания

Попиксельная потеря MSE записывается как:

$$l_{MSE}^{SR} = \frac{1}{r^2 W H} \sum_{x=1}^{rW} \sum_{y=1}^{rH} (I_{x,y}^{HR} - G_{\theta_G}(I^{LR})_{x,y})^2 \quad (4)$$

Это наиболее часто использующаяся задача оптимизации для SR изображения во многих современных подходах. Однако, при достижении особенно высокого PSNR, решениям на основе MSE оптимизации часто не хватает высокочастотных компонент, и это приводит к неудовлетворительному решению с точки зрения восприятия, оно содержит чрезмерно гладкие текстуры (см. Рисунок 1).

За место того чтобы использовать на попиксельные потери мы будем опираться на идеи Gatys et al. (Style-transfer), Bruna et al. и Johnson et al. и использовать функцию потерь, которая ближе к сходству восприятия. Мы определим VGG потерю, основанную на слоях активации ReLU предобученной 19-ти слойной VGG модели описанной Simonyan и Zisserman. Под  $\phi_{i,j}$  мы будем полагать карту особенностей, полученную на  $j$ -ом слое свёртки (после активации) перед  $i$ -ым слоем максимального пулинга в сети VGG19, которые мы считаем данными. Определим VGG потерю как евклидово расстояние между представлениями особенностей восстановленного изображения  $G_{\theta_G}(I^{LR})$  и опорного изображения  $I^{HR}$ :

$$l_{VGG/i.j}^{SR} = \frac{1}{W_{i,j}H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} (\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}(G_{\theta_G}(I^{LR}))_{x,y})^2 \quad (5)$$

Здесь  $W_{i,j}$  и  $H_{i,j}$  описывают размерности соответствующих карт особенностей в сети VGG.

### Состязательная потеря

В добавок к потерям содержания, описанным выше, мы также добавляем генеративную компоненту в наш GAN для потери восприятия. Это побуждает нашу сеть отдавать предпочтение решениям, которые лежат во множестве естественных изображений, чтобы обмануть дискриминирующую сеть. Генеративные потери  $l_{Gen}^{SR}$  определены на вероятностях дискриминатора  $D_{\theta_D}(G_{\theta_G}(I^{LR}))$  по всем обучающим образцам:

$$l_{Gen}^{SR} = \sum_{n=1}^N -\log D_{\theta_D}(G_{\theta_G}(I^{LR})) \quad (6)$$

Здесь  $D_{\theta_D}(G_{\theta_G}(I^{LR}))$  это вероятность, что восстановленное изображение  $G_{\theta_G}(I^{LR})$  это естественное HR изображение. Для лучшего градиентного подхода мы минимизируем  $-\log D_{\theta_D}(G_{\theta_G}(I^{LR}))$ , вместо  $\log[1 - D_{\theta_D}(G_{\theta_G}(I^{LR}))]$ .

## 5.3. Эксперименты

### Данные и измерения сходства

Мы производили эксперименты на трех широко используемых эталонных наборах данных Set5, Set14 и BSD100, набор тестирования BSD300. Все эксперименты производились с коэффициентом масштабирования 4x между низко- и высоко-разрешенными изображениями. Это соответствует 16x сокращению количества пикселей на изображении. Для честного сравнения, все сообщенные PSNR и SSIM измерения были высчитаны на у-канале центральной области, удалены полосы шириной 4 пикселя с каждой границы изображений, используя daala пакет (<https://github.com/xiph/daala> (commit: 8d03668)). Суперразрешенные изображения для сравнения методов включали алгоритмы ближайшего соседа, бикубический, SRCNN и SelfExSR, были взяты из онлайн материалов, дополняющих Huang et al. (<https://github.com/jbhuang0604/SelfExSR>) и для DRCN от Kim et al. (<https://cv.snu.ac.kr/research/DRCN/>). Результаты, полученные с помощью SRResNet (для потерь:  $l_{MSE}^{SR}$  и  $l_{VGG/2.2}^{SR}$ ) и варианты SRGAN доступны онлайн (<https://twitter.box.com/s/lcue6vrlrd011jktdkhhmfyk7vtjhetog>). Статические тесты были произведены в виде парных двусторонних тестов Вилкоксона со знаком ранга, и значимость определялась при  $p < 0.05$ .

Также может заинтересовать независимая разработка GAN-основанных решений на GitHub (<https://github.com/david-gpu/srez>). Однако это только экспериментальные результаты на ограниченном наборе лиц, что является более ограниченной и легкой задачей.

### Детали обучения и параметры

Обучали все сети на NVIDIA Tesla M40 GPU, используя случайные образцы из 350 тысяч изображений из ImageNet датасета. Эти изображения отличаются от тестовых изображений. Мы получили LR изображения путем уменьшения разрешения HR изображений (BGR,  $C = 3$ ), используя бикубическое ядро с масштабом  $r = 4$ . Для

каждого минибатча мы вырезаем 16 случайных  $96 \times 96$  HR подизображений из различных тренировочных изображений. Заметим, что мы можем применять генерирующую модель к изображениям любого размера, так как она полностью свёрточная. Мы отобразили пространство LR выходных изображений к отрезку  $[0, 1]$  и для HR изображений к  $[-1, 1]$ . Потеря MSE были расчитана на изображениях с яркостью в диапазоне  $[-1, 1]$ . Карты особенностей VGG были также масштабированы с коэффициентом  $1/12.75$ , чтобы получить VGG потери сравнимые по масштабу с MSE потерями. Это эквивалентно умножению Выражению 5 с масштабирующим коэффициентом  $\approx 0.006$ . Для оптимизации используется Adam с  $\beta = 0.9$ . SRResNet сети были обучены с шагом обучения  $10^{-4}$  и проведено  $10^6$  итераций. Мы используем обученную, основанную на MSE, SRResNet сеть как первоначальный вариант для генератора, при обучении действительного GAN, чтобы избежать нежелательных локальных минимумов. Все варианты SRGAN были обучены в  $10^5$  итераций с шагом обучения  $10^{-4}$  и другие в  $10^5$  итераций с меньшим шагом  $10^{-5}$ . Мы чередуем обновление сети генератора и дискrimинатора, что эквивалентно  $k = 1$ , как используется в алгоритме Goodfellow et al. Наша сеть генератор имеет 16 идентичных ( $B = 16$ ) residual блоков. В течение времени тестов мы отключаем обновление Batch Normalization, чтобы получить выход, который детерминированно зависит только от входа.

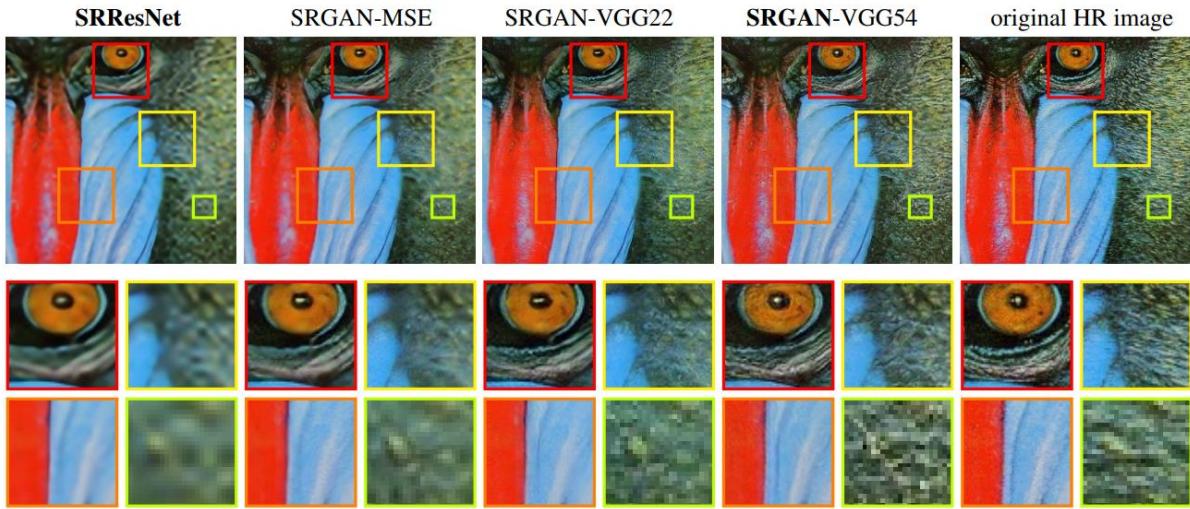
## Изучение потери содержания

Мы исследовали влияние различных вариантов потери содержания в потере восприятия для сетей на основе GAN. Конкретно мы исследуем  $l^{SR} = l_X^{SR} + 10^{-3}l_{Gen}^{SR}$  для соответствующей потери содержания  $l_X^{SR}$ :

1. **SRGAN-MSE:**  $l_{MSE}^{SR}$ , чтобы исследовать состязательную сеть со стандартной потерей содержания MSE.
2. **SRGAN-VGG22:**  $l_{VGG/2.2}^{SR}$  с  $\phi_{2.2}$ , потеря определенная на картах особенностей определяющих низкоуровневые особенности.

3. **SRGAN-VGG54**:  $l_{VGG/5.4}^{SR}$  с  $\phi_{5.4}$ , потеря определенная на картах особенностей определяющих высокоуровневые особенности из глубоких слоёв сети с большим потенциалом сфокусированным на содержания из изображений.
- Дальше эту сеть будем называть **SRGAN**.

Мы также оценивает производительность генераторной сети без состязательной компоненты для двух потерь  $l_{MSE}^{SR}$  (**SRResNet-MSE**) и  $l_{VGG/2.2}^{SR}$  (**SRResNet-VGG22**). Мы называем SRResNet-MSE как **SRResNet**. Заметим, что при обучении SRResNet-VGG22 мы добавили дополнительную общую вариационную потерю с весом  $2 * 10^{-8}$  к  $l_{VGG/2.2}^{SR}$ . Качественные результаты суммированы в Таблице 1 и визуальные примеры представлены на Рисунке 5. Даже в сочетании с состязательной потерей MSE обеспечивает решения с наибольшим PSNR, которые воспринимаются довольно гладкими и менее убедительными, чем результаты, достигнутые с компонентами потерь более чувствительными к визуальному восприятию. Это вызвано конкуренцией между потерей содержания на основе MSE и состязательной потерей. Кроме того, мы приписываем этим конкурирующим целям незначительные артефакты реконструкции, которые мы наблюдали в меньшинстве реконструкций на основе SRGAN-MSE. Мы наблюдали тренд, что используя высокоуровневые карты особенностей VGG  $\phi_{5.4}$  дает лучшую детализацию текстуры по сравнению с  $\phi_{2.2}$ (см. Рисунок 5).

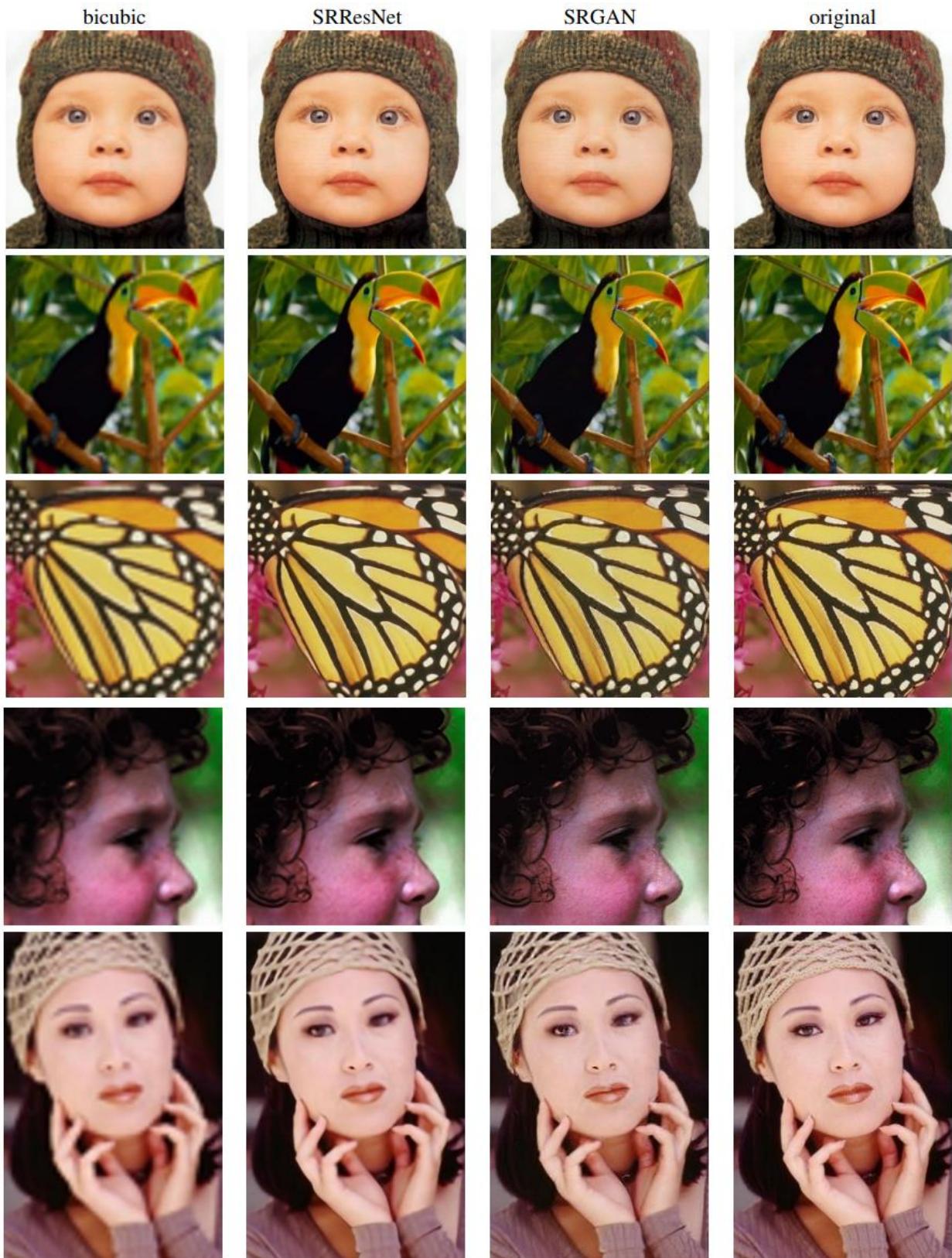


*Рисунок 9. SRResNet, SRGAN-MSE, SRGAN-VGG22, SRGAN-VGG54 результаты восстановления и соответствующее HR изображение [4x масштабирование]*

## Результаты финальных сетей

Мы сравниваем производительность **SRResNet** и **SRGAN** с методом ближайшего соседа, бикубической интерполяцией, и четырьмя современными методами. Количественные результаты подтверждают, что **SRResNet** (с точки зрения PSNR/SSIM) устанавливает новый уровень качества на трех эталонных датасетах.

Примеры суперразрешенных изображений с помощью **SRResNet** и **SRGAN** приведены на Рисунке 5. Результаты подтверждают, что SRGAN превосходит все эталонные методы с большим отрывом и устанавливает новый уровень качества для фотореалистичных изображений суперразрешения.



*Рисунок 5. Результаты из датасета Set5 используя бикубическую интерполяцию, SRResNet и SRGAN. [4x масштабирование]*

**LR изображение**



**HR изображение**



**SR изображение**



*Рисунок 6. Результаты обученной модели*

## 5.4 Выводы

В этой работе описан новый стандарт для фотогенерации суперразрешения единственного изображения, под названием **SRGAN**. Этот метод использует все современные подходы свёрточных нейронных сетей, например, residual связи, генеративно-состязательные сети для обучения, использование полученных особенностей из слоёв глубоких свёрточных сетей и т.д. Важно заметить, что генерирующая сеть научилась создавать суперразрешенные изображения, не увидев при обучении ни одного HR изображения в таком подходе, таким образом она сама научилась выделять определенные текстуры и цвета для специального восстановления определенных областей. Также подход с функцией потери содержания подчеркивает все ограничения PSNR-сфокусированных подходов и это явно подтверждает визуальное качество, которое показывает явные преимущества **SRGAN** подхода в создании фотогенерации суперразрешения.

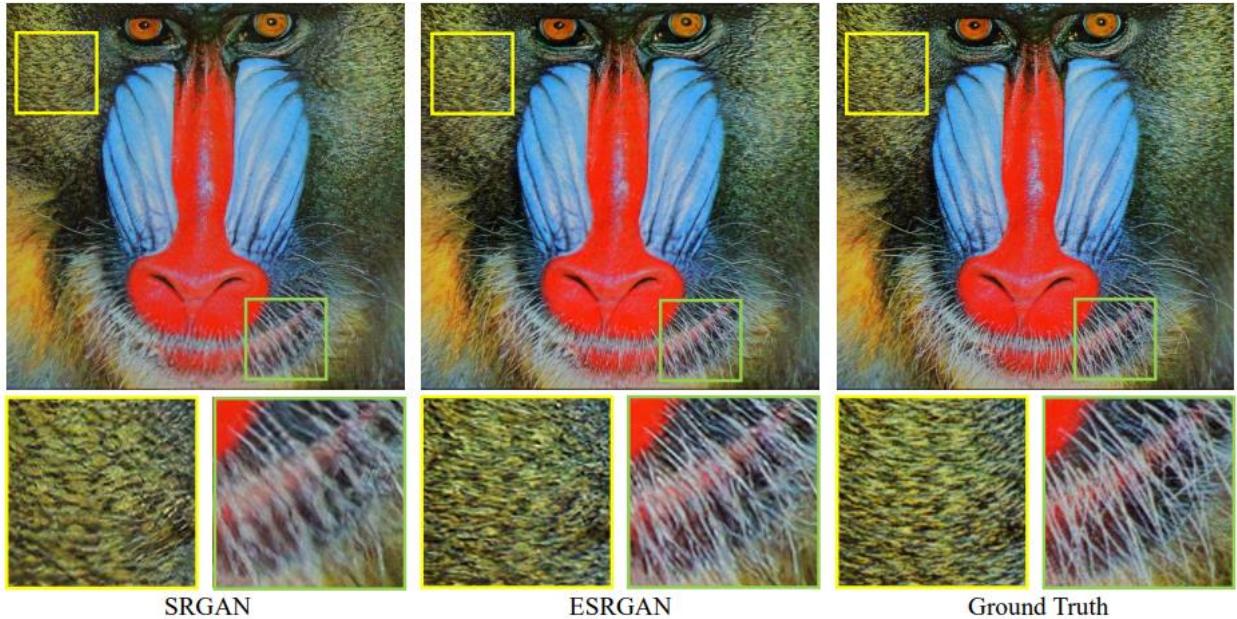
# **6. Усовершенствованные генеративно-состязательные сети супер-разрешения**

## **6.1 Введение**

Супер-разрешение одного изображения (SISR), как фундаментальная низкоуровневая проблема компьютерного зрения, привлекает все больше внимания исследовательского общества и AI компаний. SISR нацелен на восстановление изображения высокого разрешения (HR) из одного изображения низкого разрешения (LR). С тех пор как первая работа по SRCNN, предложенная Dong et al., подходы с глубокими свёрточными нейронными сетями (CNN) принесли стремительное развитие. Различные архитектуры сетей и стратегии обучения постоянно улучшают SR результаты, особенно значение пикового отношения сигнал / шум (PSNR). Однако, эти PSNR-ориентированные подходы имеют тенденцию давать чрезмерно сглаженные результаты без важных высокочастотных деталей, поскольку метрика PSNR фундаментально не соответствует субъективной оценке человеческого восприятия.

Несколько методов, основанных на восприятии, были предложены для улучшения визуального качества SR результатов. Например, потеря восприятия, предложенная для оптимизации супер-разрешенных моделей в пространстве признаков вместо пространства пикселей. Генеративно состязательные сети используются в SR, чтобы побудить сеть отдавать предпочтение решениям, которые более похожи на естественные изображения. Первоочередные семантические особенности изображений дополнительно включены, чтобы улучшить восстановление деталей текстуры. Один из главных этапов в достижении визуально приятных результатов это - SRGAN. Основная модель построена на residual блоках и оптимизирована с использованием потери восприятия в рамках GAN обучения. Благодаря этим техникам, SRGAN значительно улучшает общее визуальное качество по сравнению с PSNR-ориентированными методами.

Однако, до сих пор существует явный разрыв между SRGAN результатами и истинными изображениями, показанный на Рисунке 1. В этом исследовании вновь рассматриваются ключевые компоненты SRGAN и улучшается модель в трех аспектах. Во-первых, улучшается структура сети, внедряется блок Residual-in-Residual Dense Block (RDDB), который обладает большей емкостью и проще в обучении. Также удаляются слои Batch Normalization (BN), и используется residual scaling, и меньшая инициализация, чтобы облегчить обучение очень глубокой сети. Во-вторых, улучшается дискриминатор с помощью Relativistic average GAN (RaGAN), который учится судить "является ли это изображение реальным или поддельным". Эксперименты показали, что эти улучшения помогают генератору восстановить более реалистичные детали текстуры. В-третьих, предлагаются улучшения потери восприятия за счет использования VGG особенностей перед активацией, за место после активации, как в SRGAN. Эмпирически получено, что скорректированная потеря восприятия обеспечивает более острые края и более приятные результаты, как будет показано в следующих разделах. Обширные эксперименты показывают, что улучшенный SRGAN, называемый ESRGAN, полностью превосходит все современные методы как по резкости, так и по деталям (см. Рисунок 1).

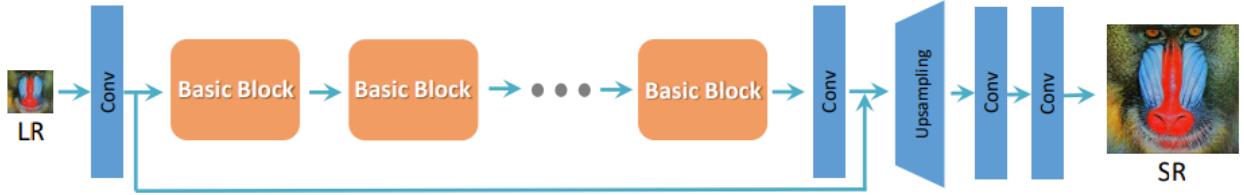


*Рисунок 1. Результаты супер-разрешения x4 для SRGAN, предложенный ESRGAN и реальное изображение. ESRGAN превосходит SRGAN в резкости и деталях.*

Чтобы сбалансировать визуальное качество и RMSE/PSNR, дальше предлагается стратегия интерполяция сети, которая может непрерывно корректировать стиль реконструкции и сглаженность. Другой альтернативой является интерполяция изображения, которая напрямую интерполирует изображение попиксельно. Этую стратегию используют для участия в регионах 1 и 2. Стратегии интерполяции сети и интерполяции изображения и их различия описаны далее.

## 6.2 Предложенный метод

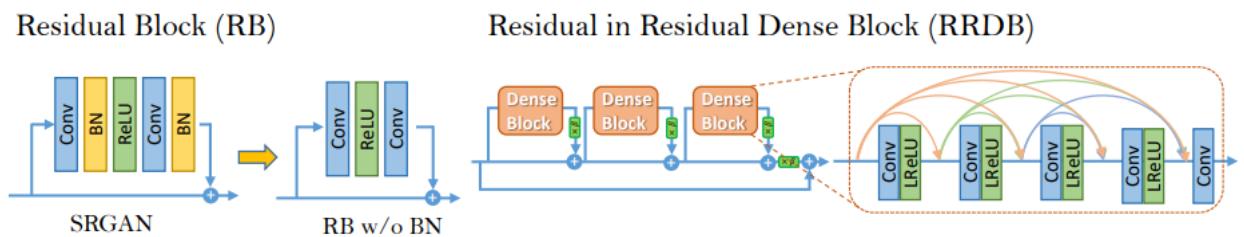
Главной целью было улучшить общее визуальное качество для SR. В этом разделе сначала опишем предложенную архитектуру сети и затем обсудим улучшения для дискриминатора и потери восприятия. И на конец, опишем стратегию интерполяции сети для баланса качества и PSNR.



*Рисунок 2. Используется стандартная архитектура SRResNet, в которой большая часть вычислений делается в пространстве особенностей LR. Можно изменить дизайн Basic Blocks (residual block, dense block, RRDB) для лучшей производительности.*

## Архитектура сети

Для дальнейшего улучшения качества восстановленного изображения с помощью SRGAN сделаем 2 модификации в структуре генератора G: 1) удалим все BN слои; 2) заменим исходный базовый блок на предложенный Residual-in-Residual Dense Block (RRDB), который объединяет многоуровневую residual сеть и dense соединения, как показано на Рисунке 3.



*Рисунок 3. Левый: Удалены BN слои из residual блока в SRGAN. Правый: RRDB блок использующийся в новой глубокой модели и  $\beta$  как residual масштабирующий параметр.*

Удаление BN слоёв повышает производительность и снижает вычислительную сложность в различных PSNR-ориентированных задачах, включая SR и уменьшение размытия (deblurring). Слои BN нормализуют особенности, используя среднее значение и дисперсию в батче во время обучения, и используют вычисленное среднее значение и дисперсию всего обучающего набора во время тестирования. Когда статистики обучающих и тестовых наборов данных сильно различаются, BN слои имеют тенденцию вносить нежелательные артифакты и ограничивают способность к обобщению.

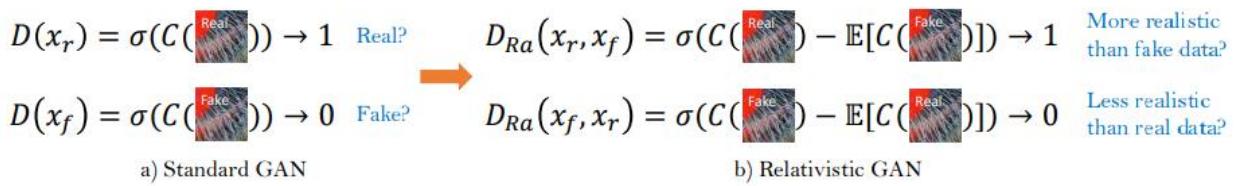
Эмпирически замечено, что BN слои с большой вероятностью приводят к созданию артифактов, когда сеть глубока и обучена в рамках GAN. Эти артифакты случайно появляются между итерациями и различными настройками, нарушая потребность в стабильной производительности по сравнению с обучением. Поэтому следует удалить BN для стабильного обучения и хорошей производительности. Кроме того, удаление BN слоёв помогает улучшить способность обобщения и уменьшить вычислительную сложность и использование памяти.

Сохраняется высокоуровневая архитектура SRGAN (см. Рисунок 3) и используются новые базовые блоки RRDB, как на Рисунке 3. Основываясь на наблюдении, что большее количество слоёв и связей может всегда повысить производительность, предлагаемый RRDB, использует более глубокую и более сложную структуру, чем оригинальный residual блок в SRGAN. В частности, как показано на Рисунке 3, предлагаемый RRDB имеет residual структуру, в которой residual обучение используется на разных уровнях. Похожая структура сети предложена в статье, обозревающей различные residual блоки (<https://arxiv.org/pdf/1608.02908.pdf>), в которых применяется многоуровневая residual сеть. Однако RRDB отличается от всех них тем, что используется dense блок в главном пути, где плотность сети становится более высокой, благодаря dense соединениям.

В дополнение к улучшенной архитектуре, мы также используем несколько методов для облегчения обучения очень глубокой сети: 1) residual scaling, т.е. уменьшение масштабов residual остатков, умножая их на константы между 0 и 1, перед добавлением их к основному пути, предотвращая нестабильность; 2) меньшая инициализация, поскольку эмпирически доказано, что residual архитектура легче обучается, когда дисперсия инициализированных параметров меньше. Детали обучения сети будут в следующей главе.

## Релятивистский дискриминатор

Помимо улучшенной структуры генератора, также улучшается дискриминатор на основе Relativistic GAN (<https://arxiv.org/pdf/1807.00734.pdf>). В отличие от стандартного дискриминатора D в SRGAN, который оценивает вероятность того, что одно входное изображение является действительным и естественным, а релятивистский дискриминатор пытается предсказать вероятность того, что реальное изображение  $x_r$  относительно более реалистичное чем поддельное  $x_f$ , как показано на Рисунке 4.



*Рисунок 4. Разница между стандартным дискриминатором и релятивистским дискриминатором.*

В частности, заменим стандартный дискриминатор на Релятивистский средний Дискриминатор (RaD = Relativistic average Discriminator), обозначенный как  $D_{Ra}$ . Стандартный дискриминатор может быть выражен в форме  $D(x) = \sigma(C(x))$ , где  $\sigma$  – сигмовидная функция и  $C(x)$  – нетрансформированный выход дискриминатора. Тогда RaD формулируется как  $D_{Ra}(x_r, x_f) = \sigma(C(x_r) - E_{x_f}[C(x_f)])$ , где  $E_{x_f}[\cdot]$  представляет операцию взятия среднего по всем поддельным данным в минибатче. Потеря дискриминатора тогда определяется как:

$$L_D^{Ra} = -\mathbb{E}_{x_r}[\log(D_{Ra}(x_r, x_f))] - \mathbb{E}_{x_f}[\log(1 - D_{Ra}(x_f, x_r))]. \quad (1)$$

Состязательные потери для генератора имеют симметричную форму:

$$L_G^{Ra} = -\mathbb{E}_{x_r}[\log(1 - D_{Ra}(x_r, x_f))] - \mathbb{E}_{x_f}[\log(D_{Ra}(x_f, x_r))], \quad (2)$$

где  $x_f = G(x_i)$  и  $x_i$  обозначает входное LR изображение. Замечено, что состязательная потеря для генератора содержит оба изображения  $x_r$  и  $x_f$ . Следовательно, генератор

извлекает выгоду из градиентов как сгенерированных данных, так и реальных данных, в то время как в SRGAN действует только сгенерированная часть. В следующей главе будет показано, что эта модификация помогает изучать более острые края и более детальные текстуры.

## Потеря восприятия

Также разработана более эффективная функция потери восприятия  $L_{percep}$ , содержащая особенности до активации, а не после, как это используется в SRGAN.

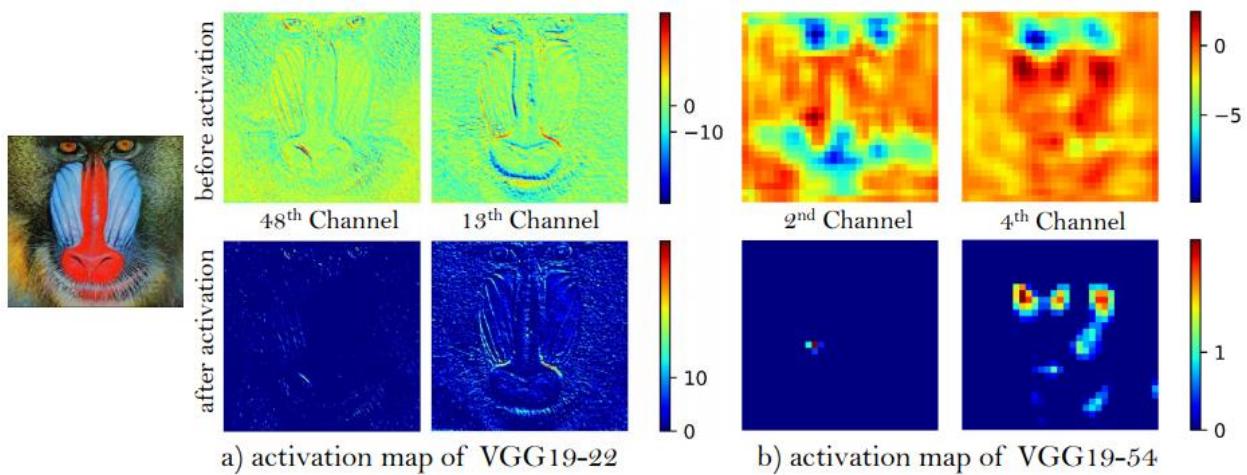
Основываясь на идее сближения со сходством восприятия, Johnson et al. предлагает потерю восприятия, и она расширена в SRGAN. Потеря восприятия ранее определена на слоях активации предобученной глубокой сети, где расстояние между двумя активированными особенностями минимизировано. Вопреки этому, предлагаем использовать особенности перед слоями активации, что позволит преодолеть два недостатка оригинального дизайна. Во-первых, активированные особенности очень прореженные, особенно после очень глубокой сети, как показано на Рисунке 5. Например, средний процент активированных нейронов для изображения "baboon" после VGG19 на 4-ом свёрточном слое составляет всего 11.17%. Разреженная активация обеспечивает слабый контроль и это приводит к снижению производительности. Во-вторых, использование особенностей после активации также приводит к неплавному восстановлению яркости по сравнению с реальным изображением, это будет показано в следующей главе.

Следовательно, общие потери генератора составляют:

$$L_G = L_{percep} + \lambda L_G^{Ra} + \eta L_1, \quad (3)$$

где  $L_1 = E_{x_i} \|G(x_i) - y\|_1$  это потеря содержания, которая оценивается с помощью  $L_1$  нормы между восстановленным изображением  $G(x_i)$  и истинным  $y$ , и  $\lambda, \eta$  – коэффициенты, чтобы балансировать различные варианты потери.

В отличие от обычно используемой потери восприятия, которая использует сеть VGG, обученную для классификации изображений, мы разработали более подходящую потерю восприятия MINC. Она основана на настроенной VGG сети для распознавания материалов, которая фокусируется на текстурах, а не на объектах. Хотя и прирост индекса восприятия, вызванный MINC потерей незначительный, будем верить, что исследование потери восприятия, которая фокусируется на текстуре, является критически важным для SR.



*Рисунок 5. Репрезентативные карты особенностей перед и после активации для изображения «baboon». По мере углубления сети большинство особенностей после активации становится неактивными, в то время как особенности перед активацией содержат больше информации.*

## Сетевая интерполяция

Чтобы устранить неприятные шумы в методах на основе GAN при сохранении хорошего качества восприятия, предлагается гибкая и эффективная стратегия - сетевая интерполяция. В частности, сначала обучим PSNR-ориентированную сеть  $G_{PSNR}$ , а затем получим сеть  $G_{GAN}$  на основе GAN путем fine-tuning. Интерполируем все соответствующие параметры этих двух сетей, чтобы получить интерполированную модель  $G_{INTERP}$ , параметры которой:

$$\theta_G^{\text{INTERP}} = (1 - \alpha) \theta_G^{\text{PSNR}} + \alpha \theta_G^{\text{GAN}}, \quad (4)$$

где  $\theta_G^{INTERP}$ ,  $\theta_G^{PSNR}$  и  $\theta_G^{GAN}$  параметры сетей  $G_{INTERP}$ ,  $G_{PSNR}$  и  $G_{GAN}$  соответственно, и  $\alpha \in [0, 1]$  параметр интерполяции.

Предлагаемая сетевая интерполяция имеет два достоинства. Во-первых, интерполированная модель способна давать значимые результаты для любого возможного  $\alpha$  без создания артефактов. Во-вторых, можно непрерывно балансировать качество восприятия и точность без переобучения модели.

Также можно рассмотреть другие методы баланса между PSNR-ориентированными и GAN-основанными методами. Например, можно напрямую интерполировать их выходные изображения (попиксельно), а не параметры сети. Однако такой подход терпит неудачу в достижении хорошего компромисса между шумом и размытием, то есть интерполированное изображение либо слишком размыто, либо с помехами из артефактов. Другой метод состоит в том, чтобы настроить веса потери содержания и потери состязания, то есть параметры лямбда и ню в Формуле 3. Но этот подход требует настройки весов потери и fine-tuning'a сети, и поэтому он слишком дорог для достижения постоянного контроля стиля изображения.

## 6.3 Эксперименты

### Детали обучения

Следуя SRGAN, все эксперименты выполняются с фактором масштаба x4 между LR и HR изображениями. Получаем LR изображение путём уменьшения разрешения HR изображения с помощью бикубической интерполяции. Размер минибатча равен 16. Разрешение обрезанного HR фрагмента составляет 128 x 128. Наблюдается, что тренировка более глубокой сети выигрывает от большего размера фрагмента, поскольку расширенное пространство помогает собирать больше семантической информации. Однако это стоит большего времени обучения и возрастают вычислительные затраты. Это явление наблюдается и в PSNR-ориентированных методах.

Процесс обучения поделен на два этапа. Во-первых, обучаем PSNR-ориентированную модель с функцией потерь  $L_1$ . Шаг обучения инициализируется как  $2 * 10^{-4}$  и уменьшается в 2 раза каждые  $2 * 10^{-5}$  изменения минибатча. Затем используем обученную PSNR-ориентированную модель в качестве инициализации генератора. Генератор обучается, используя функцию потери по Формуле 3 с  $\lambda = 5 * 10^{-3}$  и  $\eta = 1 * 10^{-2}$ . Шаг обучения установлен на  $1 * 10^{-4}$  и вдвое уменьшается на [50к, 100к, 200к, 300к] итерациях. Предобучение с попиксельной потерей помогает GAN-основанным методам получать более визуально приятные результаты. Причины состоят в том, что: 1) он может избегать нежелательных локальных минимумов для генератора; 2) после предобучения дискриминатор получает уже относительно хорошие супер-разрешенные изображения вместо предельно поддельных (черные или шумные изображения) в самом начале, что помогает ему сфокусироваться больше на распознавании текстуры.

Для оптимизации использовался Adam с параметрами  $\beta_1 = 0.9$  и  $\beta_2 = 0.999$ . Попеременно обновляем сеть генератора и дискриминатора пока модель не сойдется. Используются две настройки для генератора - одна, содержащая 16 residual блоков, с объемом сравнимым с SRGAN и другая более глубокая модель с 23-мя RRDB блоками.

## Данные

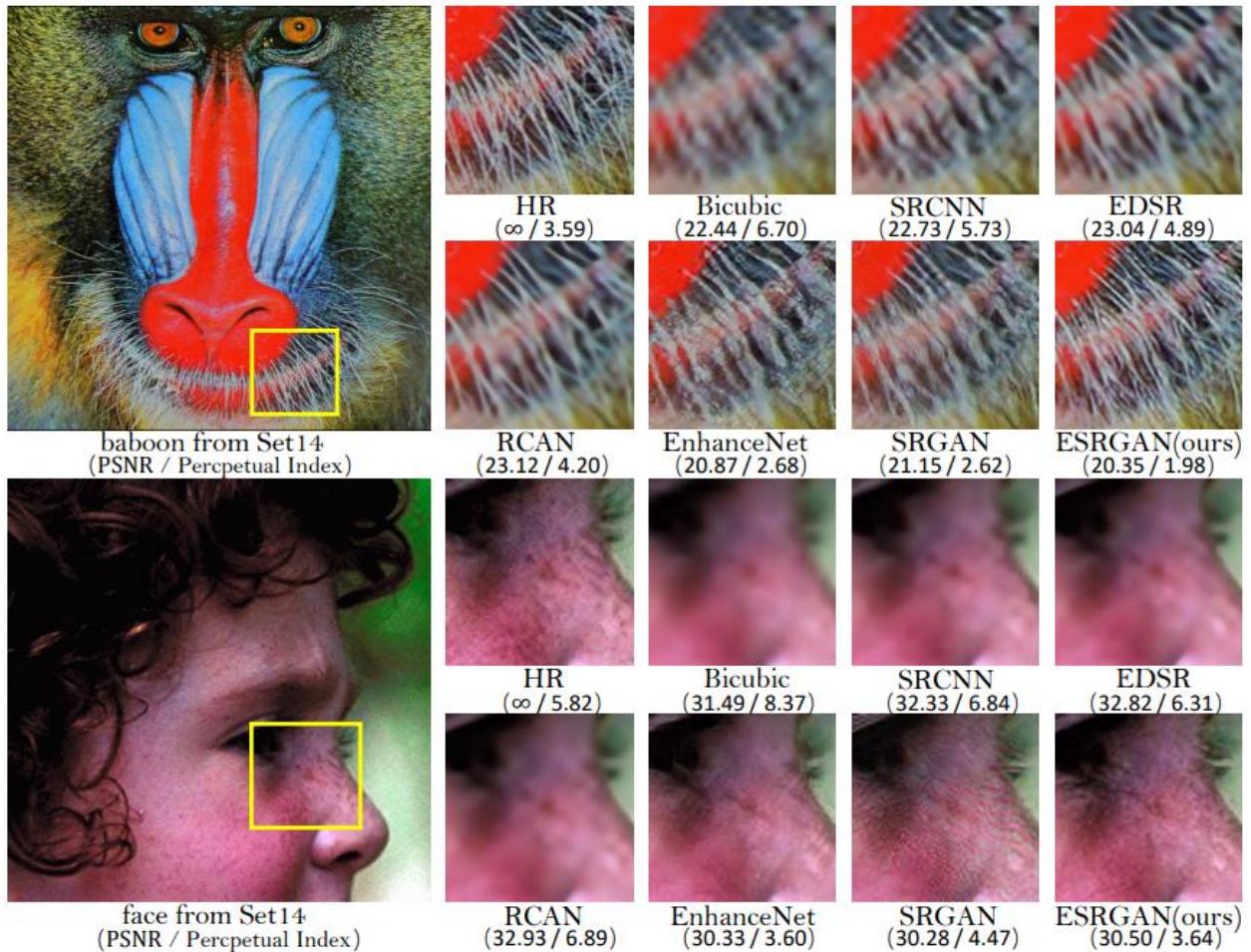
Для обучения использовался в основном датасет DIV2K, в котором изображения высокого разрешения (2K+) для задач восстановления изображений. Помимо учебного набора из DIV2K, который содержит 800 изображений, также использовались другие датасеты со сложными и разнообразными текстурами для обучения. Для этого использовались еще датасет Flickr2K, состоящий из 2650 изображений высокого разрешения 2K собранные с веб-сайта Flickr и датасет OutdoorSceneTraining(OST) для обогащения тренировочного набора. Эмпирически обнаружено, что использование этого большого датасета с различными текстурами помогает генератору производить более натуральные результаты, как показано на Рисунке 6.

Обучается модель на RGB каналах и аугментированном обучающем наборе со случайными горизонтальным отражением и 90 градусным поворотом. Модель оценивается на широко известных эталонных датасетах - Set5, Set14, BSD100, Urban100.

## **Качественные результаты**

Сравниваем конечную модель на нескольких общедоступных проверочных датасетах для современных PSNR-ориентированных методов включая SRCNN, EDSR и RCAN, а также с подходами на основе восприятия, как SRGAN и EnhanceNet. Поскольку не существует стандартизированной метрики для оценки качества восприятия, просто представим репрезентативные качественные результаты на Рисунке 6. PSNR (считается по каналу яркости в цветовом представлении YCbCr).

Как можно заметить на Рисунке 6 предложенный ESRGAN превосходит предыдущие подходы и в четкости, и в деталях. Например, ESRGAN может воспроизвести четкие и более естественные усы бабуина и текстуру травы (изображение 43074), чем PSNR методы, которые стремятся генерировать размытые результаты, и чем предыдущие методы, основанные на GAN, текстуры которых ненатуральны и содержат нежелательный шум. ESRGAN способен генерировать более детальные структуры в зданиях (изображение 102061), в то как другие методы не могут произвести достаточно деталей (SRGAN) или добавляют нежелательные текстуры (EnhanceNet). Более того, предыдущие GAN-основанные методы иногда воссоздают неприятные артифакты, например, SRGAN добавляет морщины на лицо. ESRGAN избавлен от этих артифактов и производит натуральные результаты.





*Рисунок 6. Качественные результаты ESRGAN. ESRGAN получает более натуральные текстуры, например, шерсть животных, строительные структуры и текстура травы, а также меньше нежелательных артефактов, например, артефакты на лице от SRGAN.*

## Исследование аблации

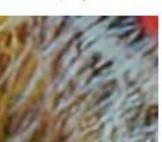
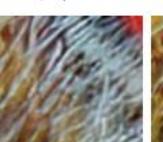
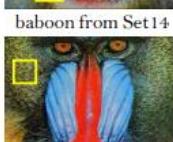
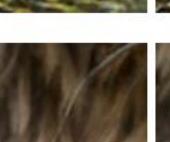
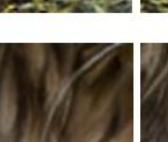
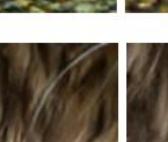
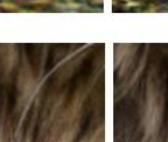
Чтобы изучить влияние каждой компоненты в предложенном ESRGAN, будем постепенно модифицировать базовую модель SRGAN и сравнивать их разницу. Общее визуальное сравнение показано на Рисунке 8. Каждый столбец представляет модель с её конфигурацией, показанной на верху. Красный знак показывает на основные изменения по сравнению с предыдущей моделью.

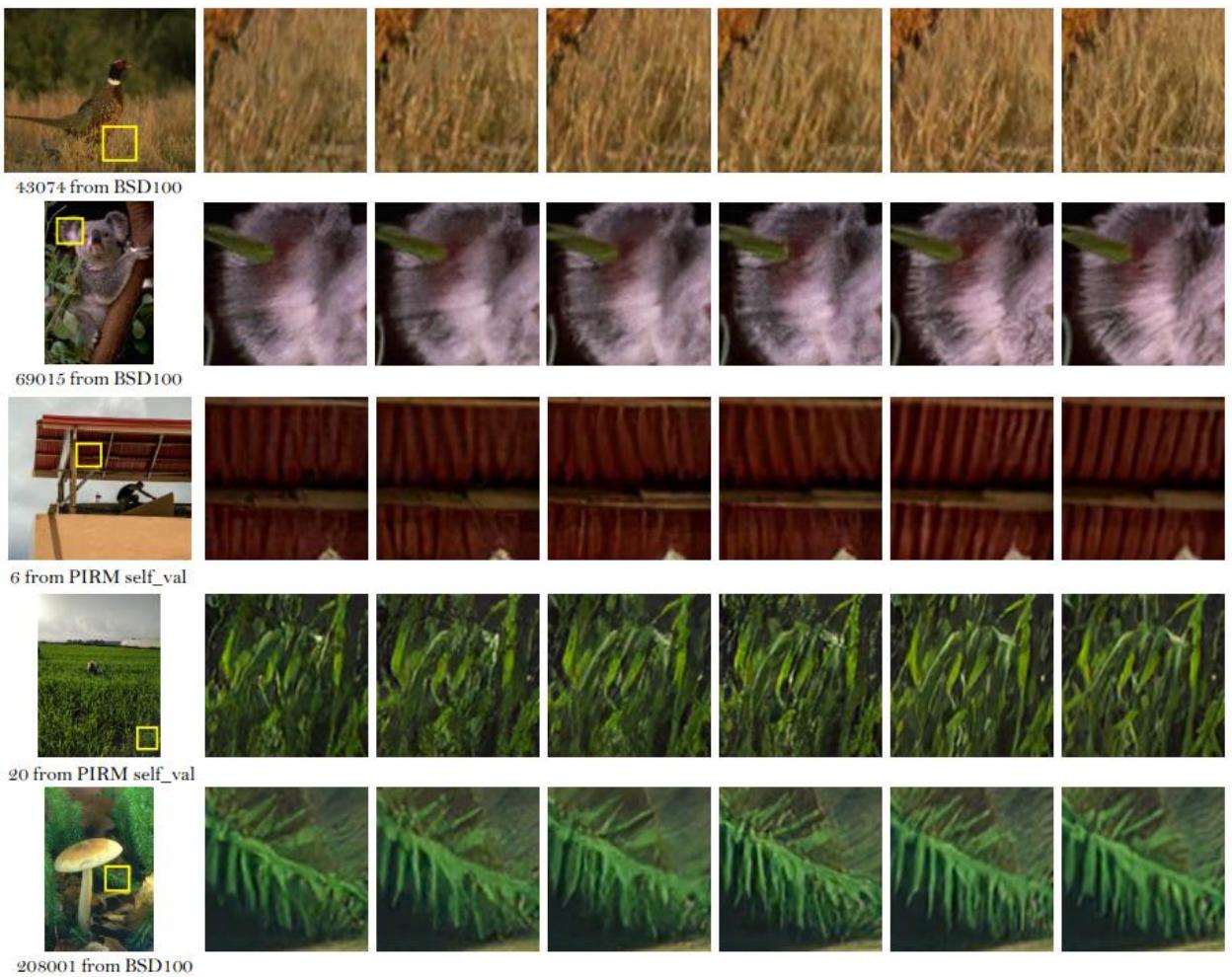
**Удаление BN.** Сначала удали все BN слои для стабильной и последовательной работы без артефактов. Это не снижает производительности, но сохраняет

вычислительные ресурсы и использование памяти. В некоторых случаях можно наблюдать небольшое улучшение из 2-го и 3-го столбцов на Рисунке 8 (например, изображение 39). Более того, если сеть более глубокая и более сложная, модель с BN с большей вероятностью внесет неприятные артефакты. Примеры показаны на Рисунке 7.



*Рисунок 7. Пример BN артефактов в GAN модели.*

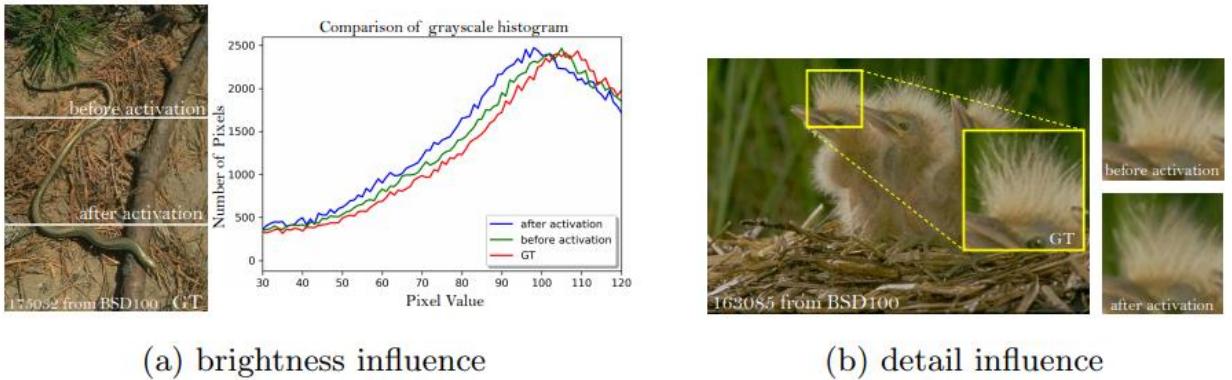
1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>	6 <sup>th</sup>	7 <sup>th</sup>
BN?	✓	✗	✗	✗	✗	✗
Activation?	After	After	Before	Before	Before	Before
GAN?	Standard GAN	Standard GAN	Standard GAN	RaGAN	RaGAN	RaGAN
Deeper with RRDB?	✗	✗	✗	✗	✓	✓
More data?	✗	✗	✗	✗	✗	✓
						
baboon from Set14						
						
baboon from Set14						
						
39 from PIRM self_val						



*Рисунок 8. Общее визуальное сравнение для демонстрации эффектов каждой компоненты в ESRGAN. Каждый столбец отображает модель с конфигурацией, описанной наверху. Красный знак указывает на ключевое улучшение по сравнению с предыдущей моделью.*

**Перед активацией в потере восприятия.** Сначала продемонстрируем, что использование особенностей перед активацией может привести к более точной яркости восстановленных изображений. Чтобы исключить влияние на текстуры и цвет, фильтруем изображение с Гауссовым ядром и строим гистограмму в оттенках серого. Рисунок 9а показывает распределение яркости для каждого случая. Использование активированных особенностей смещает распределение влево, что приводит к снижению яркости, а использование особенностей до активации приводит к более точному распределению яркости, ближе к истинному.

Дальше можно заметить, что использование особенностей перед активацией помогает создавать более острые края и более богатые текстуры, как показано на Рисунке 9b (см. перо птицы) и на Рисунке 8 (см. 3-й и 4-й столбцы), поскольку плотность особенностей перед активацией предлагает более сильный контроль, чем его может обеспечить редкие активации.



*Рисунок 9. Сравнение между взятием особенностей перед активацией или после активации.*

**RaGAN.** RaGAN использует улучшенный релятивистский дискриминатор, который, как показано, помогает изучить более острые края и более детализированные текстуры. Например, в 5-м столбце на Рисунке 8 сгенерированные изображения более четкие и с более богатыми текстурами, чем слева (см. baboon, изображение 39 и изображение 43074).

**Более глубока сеть с RRDB.** Более глубокая модель с предложенным RRDB может дополнительно улучшить восстановленные текстуры, особенно для регулярных структур, как крыша на изображении 6 на Рисунке 8, поскольку глубокая модель имеет сильную способность представления семантической информации. Также найдено, что более глубокая модель может уменьшить неприятные шумы, как на изображении 20 на Рисунке 8.

В отличие от SRGAN, который утверждал, что более глубокие модели все труднее обучать, эта глубокая модель показывает превосходную производительность с легким

обучением, благодаря вышеупомянутым усовершенствованиям, особенно предложенному RRDB без BN слоёв.

## **Сетевая интерполяция**

Сравним эффекты стратегий сетевой интерполяции и интерполяции изображения на балансирование результатов PSNR-ориентированной модели и методов на основе GAN. Применим простую линейную интерполяцию в обеих схемах. Интерполяционный параметр альфа выбирается от 0 до 1 с шагом 0.2.

Как показано на Рисунке 10, чистый GAN-основанный метод производит острые края и более богатые текстуры, но с некоторыми неприятными артефактами, в то время пока чистый PSNR-ориентированный метод выдает размытые изображения в мультишном стиле. Используя сетевую интерполяцию, неприятные артефакты уменьшаются при сохранении текстур. Напротив, интерполяция изображения не может эффективно удалить эти артефакты.

Интересно отметить, что стратегия сетевой интерполяции обеспечивает непрерывный контроль баланса между качеством восприятия и точности, как показано на Рисунке 10.



*Рисунок 10. Сравнение между сетевой интерполяцией и интерполяцией изображения.*

## 6.4 Выводы

Представлена модель ESRGAN, которая достигает неизменно лучшего качества восприятия, чем предыдущие методы SR. Сформулирована новая архитектура, содержащая несколько RDDB блоков без BN слоёв. Кроме того, полезные методы, включая residual масштабирование и меньшую инициализацию, использующиеся для облегчения обучения глубокой модели. Также предложено использование релятивистского GAN в качестве дискриминатора, который учится судить какое изображение более реалистичное, чем другое, и направляет генератор восстанавливать более детальные текстуры. Более того, улучшена потеря восприятия, с помощью использования особенностей перед активацией, которые обеспечивают более сильный контроль и таким образом восстанавливают более точную яркость и реалистичные текстуры.

# **7. Использование представления на основе формы в задаче супер-разрешения**

## **7.1 Объяснение проблемы**

В 2019 году на конференции ICLR 2019 в Новом Орлеане была представлена работа (<https://openreview.net/pdf?id=Bygh9j09KX>) о том, что считается, что сверточные нейронные сети (CNN) распознают объекты, изучая все более сложные представления форм объектов. Некоторые недавние исследования предполагают более важную роль текстур изображения. Они подвергают эти противоречивые гипотезы количественному тесту, оценивая CNN и людей-наблюдателей на изображениях с конфликтом сигналов текстуры-формы. Показывают, что нейронная сеть, обученная на ImageNet сильно склонена к распознаванию текстур, а не форм, что резко контрастирует с поведенческими данными человека и раскрывает принципиально разные стратегии классификации. Затем они демонстрируют, что та же стандартная архитектура (ResNet-50), которая изучает представление на основе текстур в ImageNet, способна вместо этого изучать представление на основе форм при обучении в Stylized-ImageNet, стилизованной версии ImageNet. Это обеспечивает гораздо лучшее соответствие поведенческим характеристикам человека в их хорошо контролируемой психофизической лабораторной обстановке (девять экспериментов на общую сумму 48 560 психофизических испытаний на 97 наблюдателях) и дает ряд неожиданных новых преимуществ, таких как улучшенные характеристики обнаружения объектов и ранее невидимая устойчивость к широкому диапазону искажений изображения, подчеркивая преимущества представления на основе формы.



(a) Texture image  
81.4% **Indian elephant**  
10.3% indri  
8.2% black swan



(b) Content image  
71.1% **tabby cat**  
17.3% grey fox  
3.3% Siamese cat



(c) Texture-shape cue conflict  
63.9% **Indian elephant**  
26.4% indri  
9.6% black swan

*Рисунок 1. Классификация стандартной сети ResNet-50 на (а) изображении текстуры (кожа слона: только текстура); (б) обычное изображение кота (вместе с формами и текстурами), и (с) изображение с конфликтом текстурной формы, сгенерированное путём передачи стиля между первыми двумя изображениями.*

## 7.2 Выводы авторов

Они предоставили доказательства того, что в настоящее время машинное распознавание чрезмерно зависит от текстур объектов, а не от глобальных форм объектов, как принято считать. Продемонстрировали преимущества представления на основе формы для надежного вывода (используя набор данных Stylized-ImageNet для создания такого представления в нейронных сетях). Они предполагают, что выводы, общедоступные весовые коэффициенты моделей, код и набор поведенческих данных (49 тыс. испытаний на 97 наблюдателях) помогут для достижения трех целей:

1. Улучшенное понимание представлений и предубеждений CNN.
2. Шаг к более правдоподобным моделям распознавания визуальных объектов человека.
3. Полезная отправная точка для будущих начинаний, когда знание предметной области предполагает, что представление на основе формы может быть более выгодным, чем представление на основе текстуры.

## 7.3 Использование в супер-разрешении

В архитектуре сети супер-разрешения ESRGAN обучение происходит в 2 стадий: сначала обучается сеть с функцией ошибки MSE, а затем обучается на ошибке, состоящей из 3-х частей:  $L_1$  – потеря содержания,  $L_G^{Ra}$  – состязательная потеря и  $L_{percep}$  – потеря восприятия. В последней используется сеть VGG, в которой пропускается супер-разрешенное и оригинальное изображения. После этого высчитывается MSE ошибка между промежуточными свёрточными слоями, используя карты особенностей перед активациями.

Так как сеть, обученная на стилизованных изображениях из датасета Stylized-ImageNet, стремиться выделить из изображения формы различных структур и объектов, которые так необходимы в задаче супер-разрешения для восстановления необходимых форм, а нужные текстуры и так содержаться в исходном изображении.

Исходя из этих соображений я в своей работе подменяю на втором этапе обучения ESRGAN сеть для потери восприятия, вместо VGG использую предобученный ResNet50 на Stylized-ImageNet датасете.

## 8. Заключение

В этой работе полностью исследована задача супер-разрешения одного изображения с помощью нейронных сетей, с момента появления этой задачи до state-of-the-art (самых современных) архитектур моделей.

## 9. ИСТОЧНИКИ

1. Image Super-Resolution Using Deep Convolutional Networks  
[<https://arxiv.org/pdf/1501.00092.pdf>]
2. Fast and Accurate Image Super-Resolution with Deep Laplacian Pyramid Networks  
[<http://vllab.ucmerced.edu/wlai24/LapSRN/>]
3. Keras: The Python Deep Learning library [<https://keras.io/>]
4. Single-Image Super-Resolution: A Benchmark [<https://pdfs.semanticscholar.org/a286/af401232dcf181af6790873d92585a85f370.pdf>]
5. Super-resolution imaging [[https://en.wikipedia.org/wiki/Super-resolution\\_imaging](https://en.wikipedia.org/wiki/Super-resolution_imaging)]
6. Review: SRCNN (Super Resolution) [<https://medium.com/coinmonks/review-srcnn-super-resolution-3cb3a4f67a7c>]
7. Network In Network [<https://arxiv.org/pdf/1312.4400.pdf>]
8. Generative Adversarial Networks, Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, 2014 [<https://arxiv.org/pdf/1406.2661.pdf>]
9. Генеративно состязательная сеть [[https://ru.wikipedia.org/wiki/Генеративно-состязательная\\_сеть](https://ru.wikipedia.org/wiki/Генеративно-состязательная_сеть)]
10. Generative adversarial network [[https://en.wikipedia.org/wiki/Generative\\_adversarial\\_network](https://en.wikipedia.org/wiki/Generative_adversarial_network)]
11. Introduction to Deep Neural Networks (Deep Learning)  
[<https://deeplearning4j.org/neuralnet-overview.html>]
12. How to Train a GAN? Tips and tricks to make GANs work  
[<https://github.com/soumith/ganhacks>]
13. GAN: A Beginner's Guide to Generative Adversarial Networks  
[<https://deeplearning4j.org/generative-adversarial-network>]
14. Generative Learning algorithms, Andrew Ng's Stanford notes [<http://cs229.stanford.edu/notes/cs229-notes2.pdf>]
15. On Discriminative vs. Generative classifiers: A comparison of logistic regression and naïve Bayes [<http://papers.nips.cc/paper/2020-on-discriminative-vs-generative-classifiers-a-comparison-of-logistic-regression-and-naive-bayes.pdf>]
16. From GAN to WGAN

[<https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html>]

17. Generative adversarial networks

[<https://habr.com/post/352794/>]

18. Up-sampling with Transposed Convolution

[<https://towardsdatascience.com/up-sampling-with-transposed-convolution-9ae4f2df52d0>]

19. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network [<https://arxiv.org/pdf/1609.04802.pdf>]

20. Keras-GAN [<https://github.com/eriklindernoren/Keras-GAN>]

21. SRGAN, a TensorFlow Implementation

[<https://towardsdatascience.com/srgan-a-tensorflow-implementation-49b959267c60>]

22. Is the deconvolution layer the same as a convolutional layer?

[<https://arxiv.org/ftp/arxiv/papers/1609/1609.07009.pdf>]

23. ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks

[<https://arxiv.org/pdf/1809.00219.pdf>]

24. Гудфеллоу Я., Бенджио И., Курвилль А. Глубокое обучение / пер. с анг. А. А. Слинкина. – 2-е изд., испр. – М.: ДМК Пресс, 2018.