

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	14
1.2 Современные методы	14
2. Супер-разрешения с помощью свёрточных нейронных сетей	14
2.1 Введение	14
2.2 Описание метода	18
Извлечение и представление патчей	20
Нелинейное отображение	20
Реконструкция	21
2.3 Обучение	22
2.4 Эксперименты	23
2.5 Заключение	28
3. Генеративно-состязательные сети	29
3.1 Введение	29
3.2 Главная идея	31
3.3 Алгоритм обучения	31
3.4 Эксперименты	33
3.5 Преимущества и недостатки	35
4. Практическое использование генеративно-состязательных сетей	36
4.1 Введение	36
4.2 Модель	37
4.3 Задача нахождения параметров нормального распределения	40
4.4 Задача приближения смеси нормальных распределений	45
4.5 Задача приближения распределения изображений	49

4.6 Итоги.....	52
5. Супер-разрешения с помощью свёрточных генеративно-состязательных сетей.....	53
5.1 Введение	53
5.2 Метод.....	55
Архитектура состязательной сети	56
Функция потери восприятия.....	58
5.3. Эксперименты	61
Данные и измерения сходства	61
Детали обучения и параметры.....	62
Тестирование средней оценки мнения (Mean opinion score = MOS).....	63
Изучение потери содержания	66
Результаты финальных сетей	67
5.4 Возможные улучшения.....	70
5.5 Выводы	71
6. Усовершенствованные генеративно-состязательные сети супер-разрешения	72
6.1 Введение	72
6.2 Предложенный метод	76
Архитектура сети	76
Релятивистский дискриминатор.....	78
Потеря восприятия	80
Сетевая интерполяция.....	82
6.3 Эксперименты	83

Детали обучения.....	83
Данные	84
Качественные результаты.....	85
Исследование абляции	87
Сетевая интерполяция.....	92
6.4 Выводы	93
7. Заключение.....	94
8. Источники	94

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 даёт 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 налицоует очень глубокое, приближающееся к человеческому понимание 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.

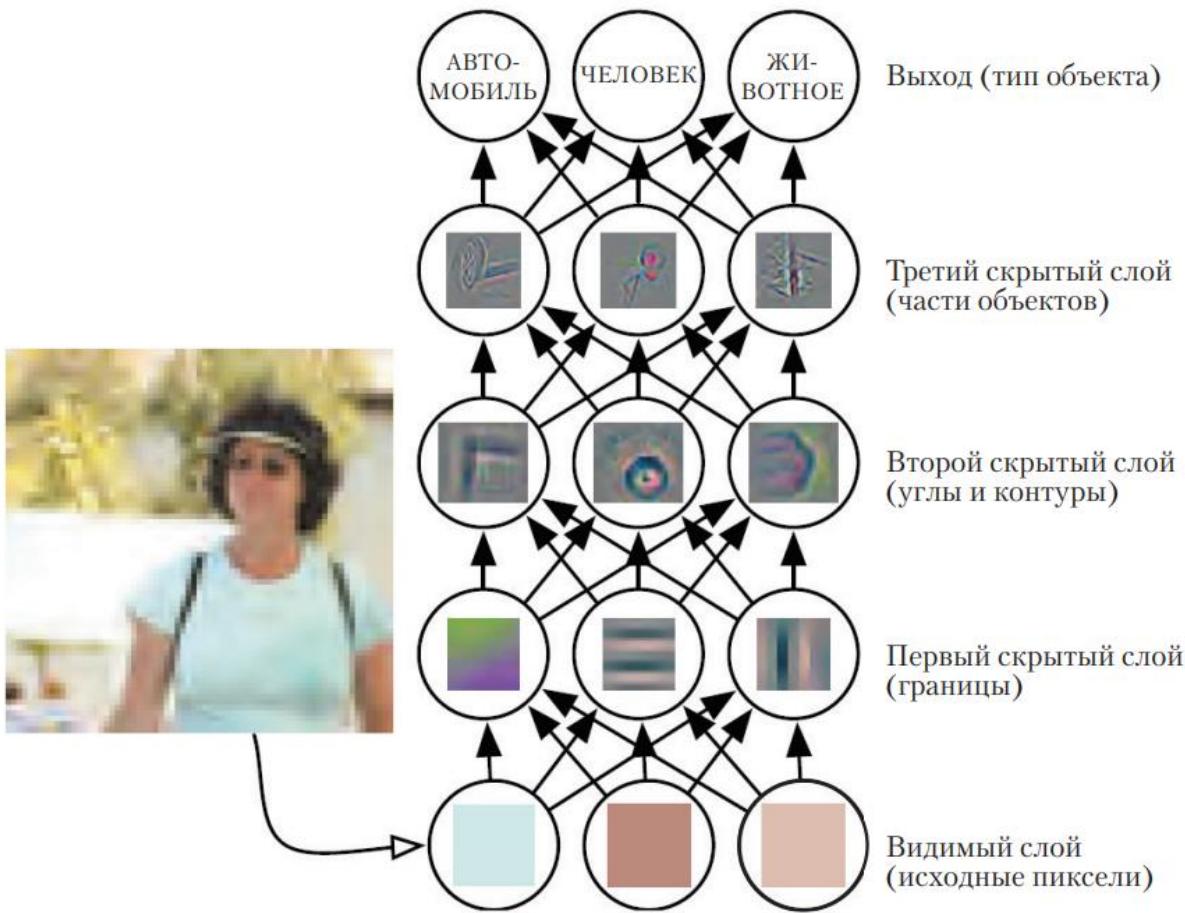


Рисунок 1. Иллюстрация модели глубокого обучения.

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 домини-рующих 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 почерпнуть 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

1.2 Современные методы

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 использую 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 предоб-рабатываются (например, вычитается среднее и нормализуются). Эти патчи за-тем кодируются словарём более низкого разрешения. Разреженные коэффици-енты передаются в словарь с высоким разрешением для восстановления пат-чей более высокого разрешения. Перекрывающиеся фрагменты агрегируют (например, взвешенное среднее) для получения окончательного результата. Такая последовательность действий используется большинством методов, ос-нованных на примерах, которые выделяют особое внимание оптимизации сло-варей или созданию эффективных функций отображения, однако остальные шаги таких подходов редко можно оптимизировать.

В данной работе хотелось показать, что такой сложный алгоритм из не-скольких 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 прин-ципиально отличается от существующих подходов, основанных на примерах тем, что он не явно изучает словари или многообразие для моделирования про-странства 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 Su-per-resolution полностью получен с помощью обучения с пред- и постобработ-кой.

Авторы этого метода назвали его Super-Resolution Convolutional Neural Network (SRCNN). Предлагаемый SRCNN обладает несколькими 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.

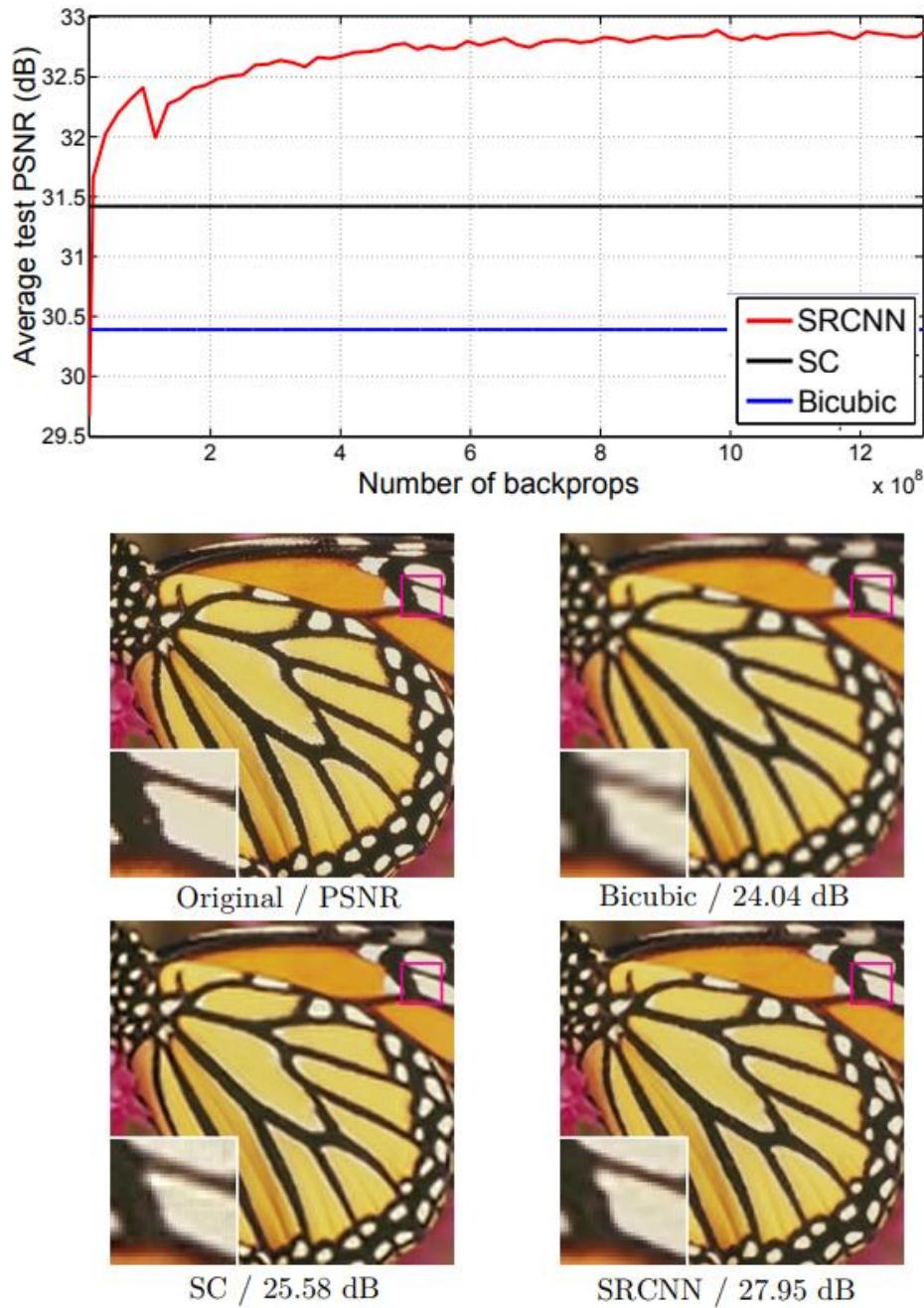


Рисунок 1. Сравнение SRCNN с популярными алгоритмами

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:

1. **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.
2. **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.
3. **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.

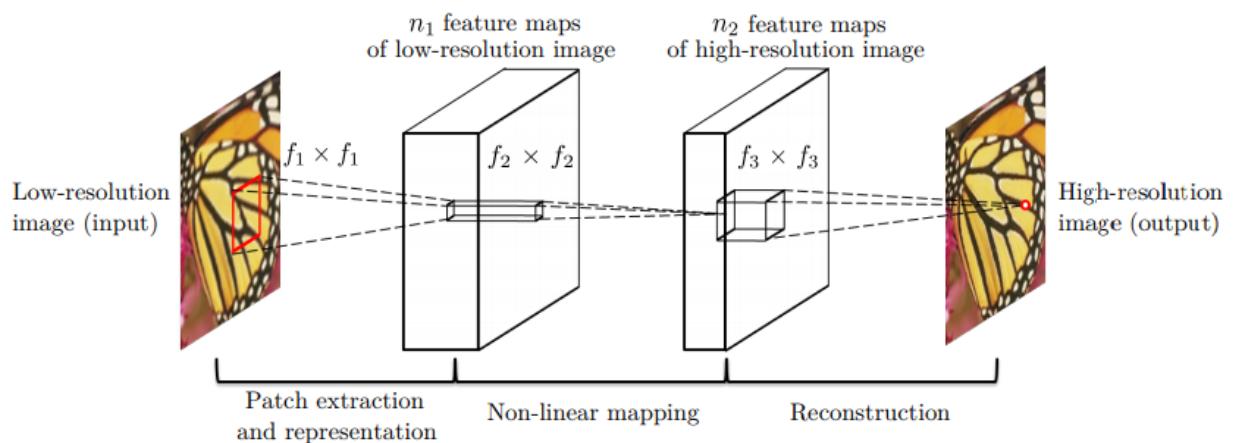


Рисунок 2. Структура SRCNN

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 1×1 . Эта интерпретация справедлива только для фильтров размерности 1×1 . Но это можно легко обобщить на фильтры больших размеров, например, 3×3 или 5×5 . В этом случае нелинейное

отображение находится не на патче из входного изображения, а на 3x3 или 5x5 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 перекрывающиеся 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 c 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 мотивированы разными представлениями, однако они приводят к одной и той же форме, что и свёрточные слои. Объединим 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_{sub} , f_{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 $((f_{\text{sub}} - f_1 - f_2 - f_3 + 3)^2, c)$. Функция потери MSE оценивается только по разнице между центральными пикселями исходного изображения и выходом сети. Хотя 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 увеличивали

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 ~ 11 т. 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): $f1=9$, $f2=3$, $f3=5$, $n1=64$, $n2=32$.

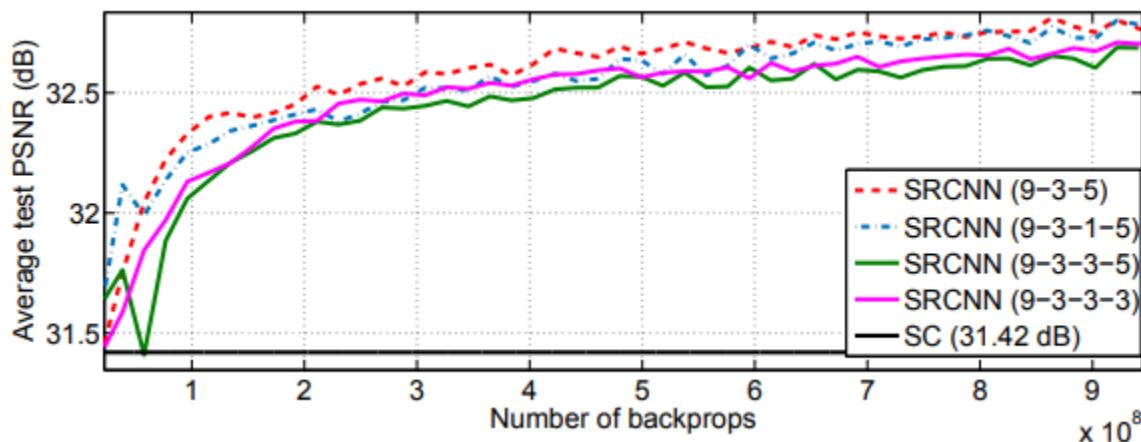


Рисунок 3. Исследование размеров сети

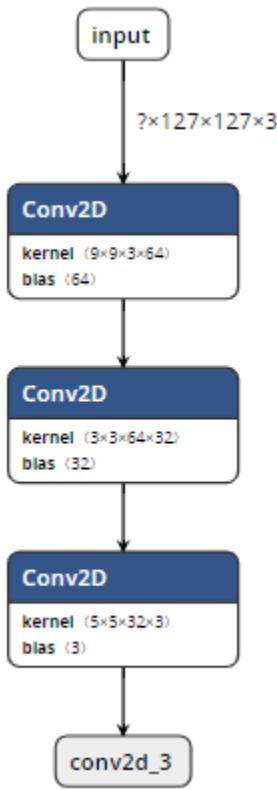


Рисунок 4. Структура сети

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 преимуще-

ство SRCNN показывает, когда сеть достаточно обучилась, уже пройдя $3 * 10^8$ шагов обучения, сеть превосходит результаты всех основных методов суперразрешения.

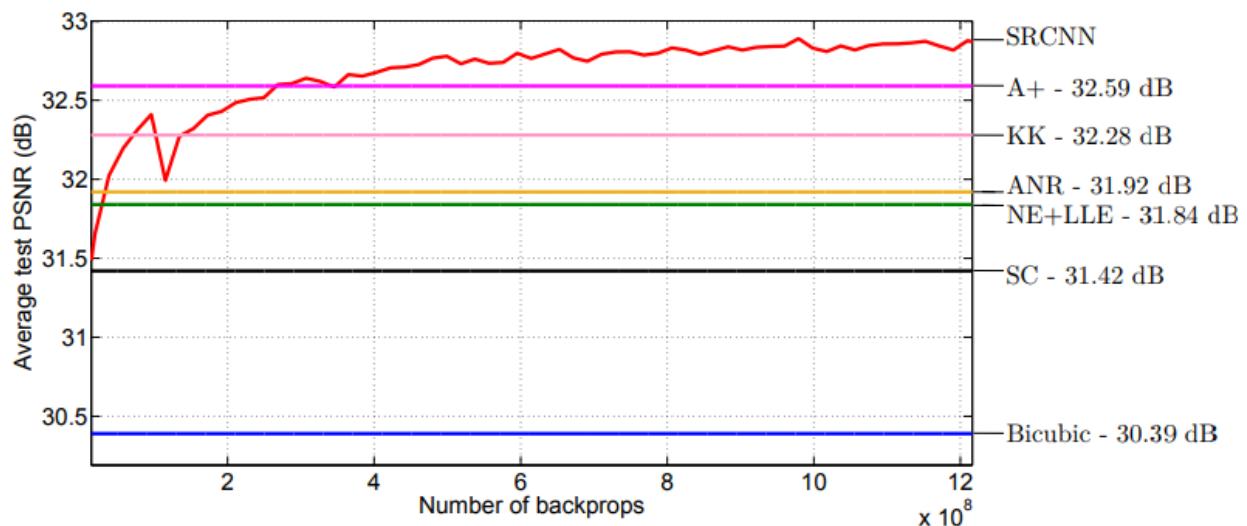


Рисунок 5. Сравнение SRCNN с другими популярными методами SR

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.

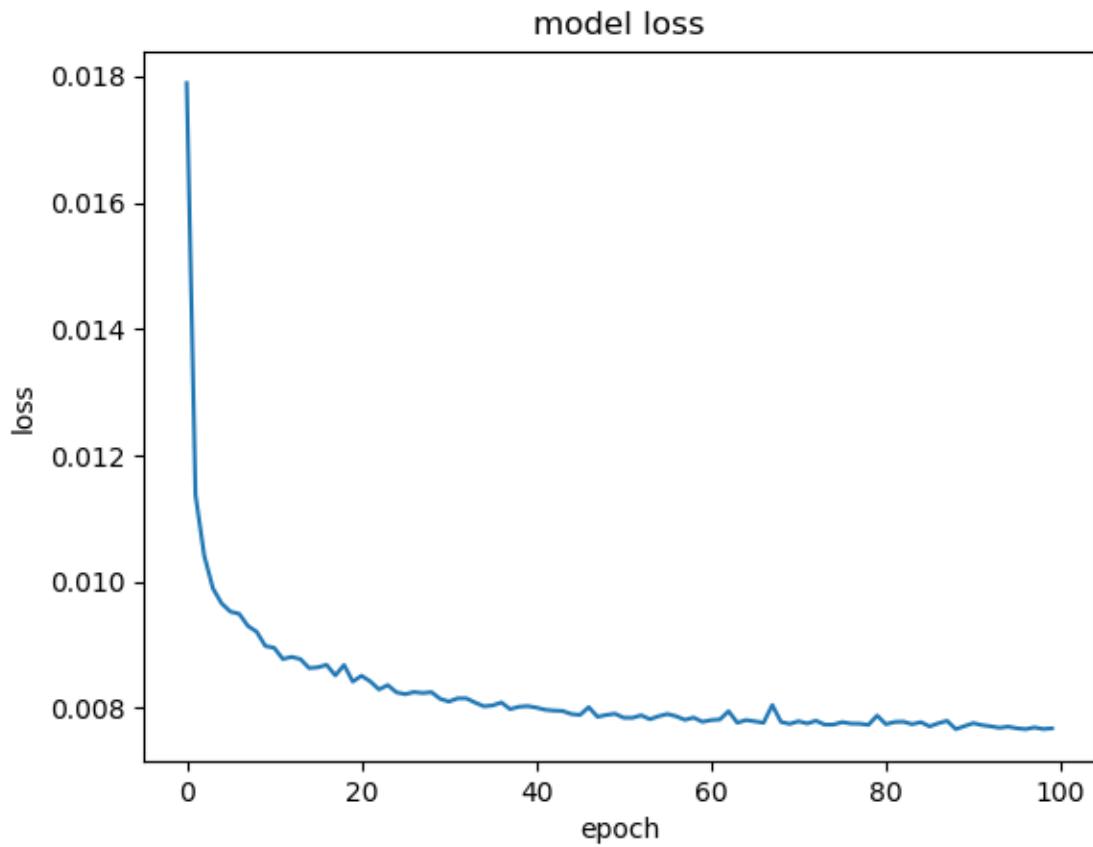
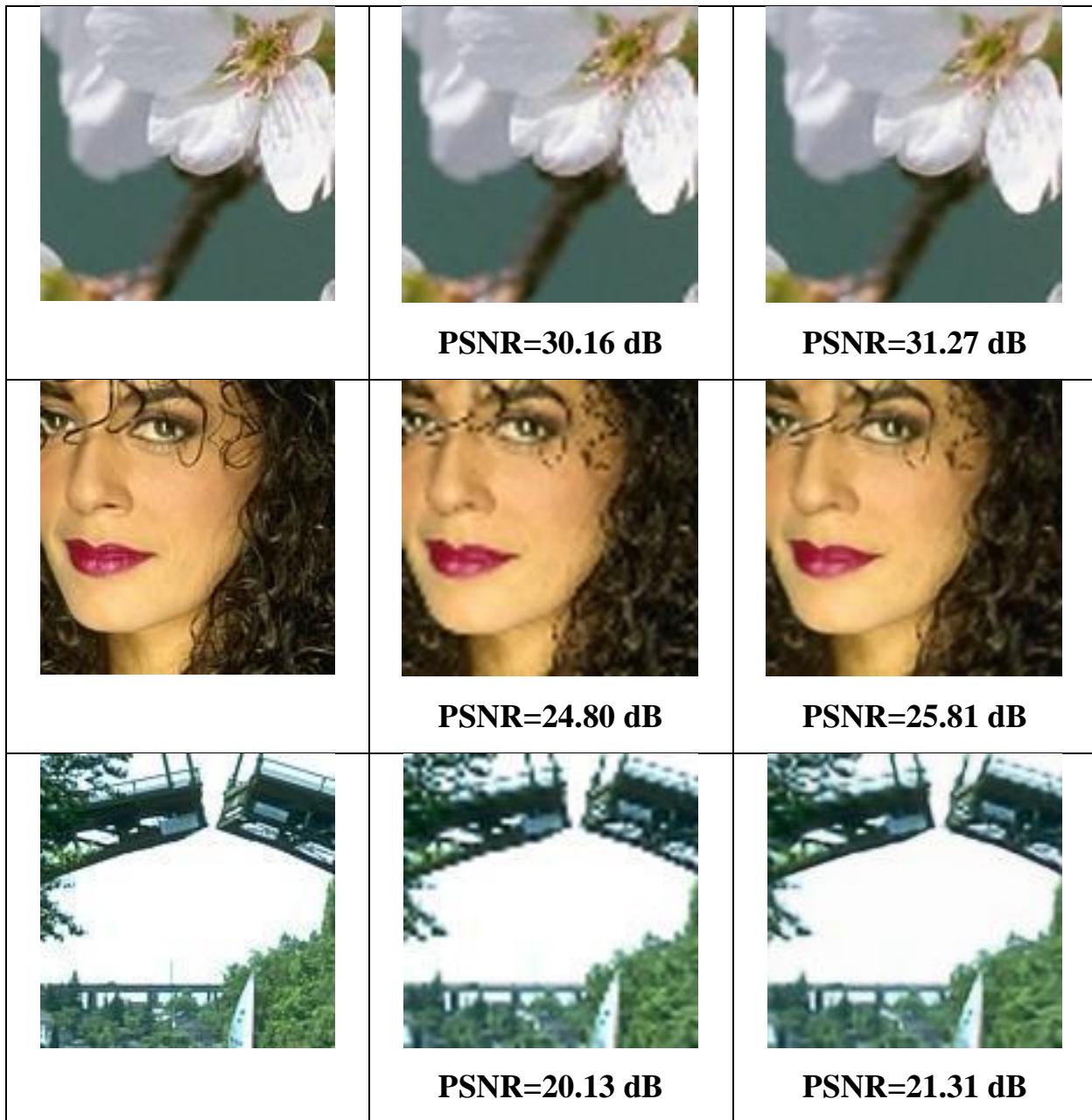


Рисунок 6. Ошибка на обучающем наборе при обучении

Рассмотрим также несколько примеров получившихся картинок:

Исходная картинка высокого разрешения	Бикубическая интерполяция	SRCNN

PSNR=20.28 dB PSNR=22.55 dB



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 многомер-

ные 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 θ_g . Мы также определяем вторую нейронную сеть $D(x, \theta_d)$, 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_{data} , если ему была предоставлена

лена 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 $p_g = p_{data}$, 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 дискриминатору, 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  $p_{data}(\mathbf{x})$ .
        • 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 σ 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 ± 2	1909 ± 66
Stacked CAE [3]	121 ± 1.6	2110 ± 50
Deep GSN [6]	214 ± 1.1	1890 ± 29
Adversarial nets	225 ± 2	2057 ± 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 σ , выбранным по проверочной выборке каждого сета. В TFD σ 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.

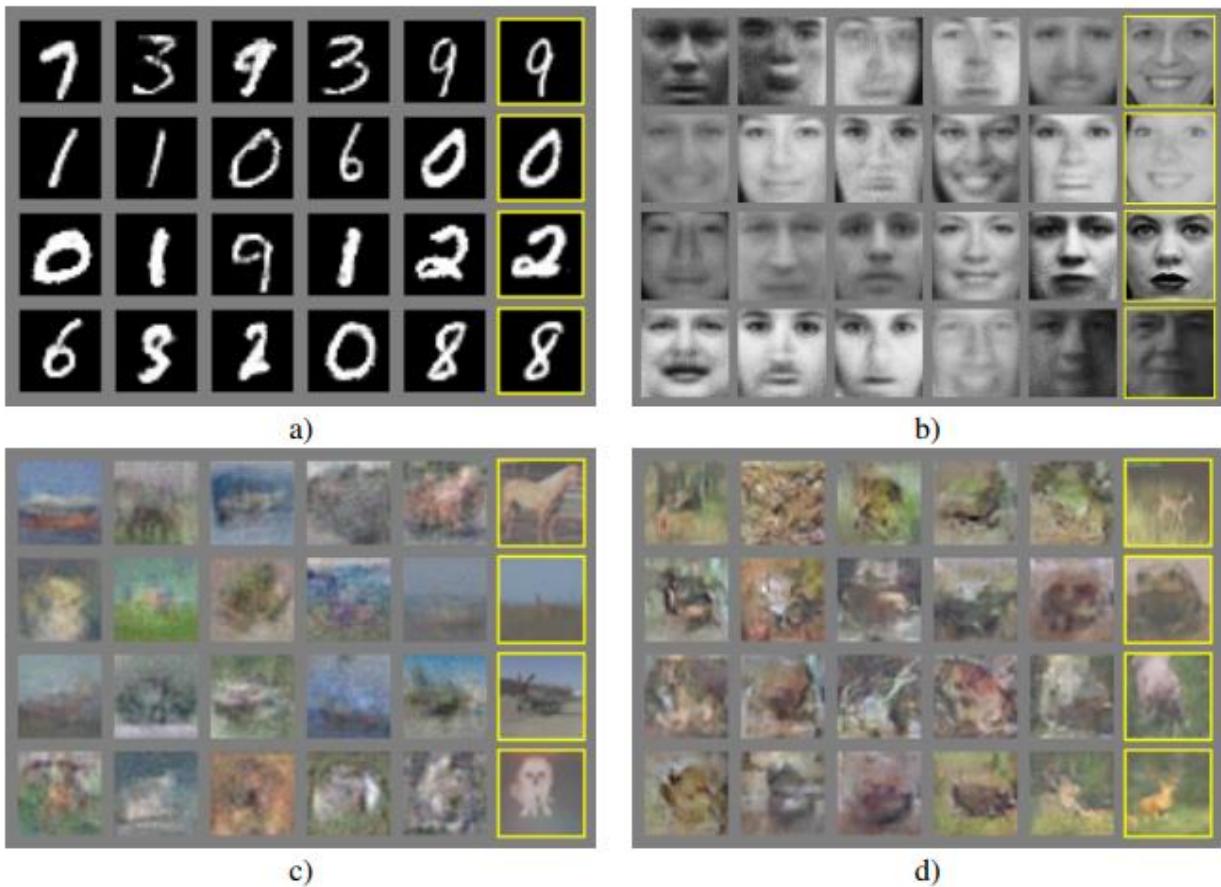


Рисунок 1: Визуализация образцов из модели. В самом правом столбце показан ближайший пример обучения из соседних образцов, чтобы показать, что модель не запомнила сет обучения. Образцы — это честные случайные розыгрыши, а не хорошо выбранные. В отличии от большинства визуализаций глубинных генеративных моделей, эти изображения показывают фактические образцы из модели распределений, а

не условные образы скрытых объектов. Более того, эти образы некоррелированные, так как процесс отбора образцов не зависит от цепного смещивания Маркова. а) MNIST б) TFD в) CIFAR-10 (полносвязная модель) д) 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 Ω 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 при-

мера 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.

5. 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.

```
mean_train = 2.5
stddev_train = 3.25
train_data = np.random.normal(mean_train, stddev_train, size=(BATCH_SIZE, 1))
```

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

```
generator_input = tf.random_normal(shape=(BATCH_SIZE, 1), mean=0.0, stddev=1.0)
def generator(generator_input):
    with tf.variable_scope('generator_params', reuse=tf.AUTO_REUSE):
        mean = tf.get_variable(
            name="mean",
            initializer=tf.constant_initializer(0.0),
            shape=[1])
        stddev = tf.get_variable(
            name="stddev",
            initializer=tf.constant_initializer(1.0),
            shape=[1])
    return generator_input * stddev + mean
```

Теперь определим дискриминатор. Он будет полно связной нейронной сетью, например, с 3 скрытыми слоями.

```
input_data = tf.placeholder(dtype=tf.float32, shape=(BATCH_SIZE, 1))
def discriminator(input_data):
    result = input_data
    features = 1
    with tf.variable_scope('discriminator_params', reuse=tf.AUTO_REUSE):
        for i in range(HIDDEN_LAYERS_NUMBER):
            weights = tf.get_variable(
                "weights_%d" % i,
                initializer=tf.truncated_normal_initializer(stddev=0.1),
                shape=[features, HIDDEN_LAYER_SIZE]
            )
            biases = tf.get_variable(
                "biases_%d" % i,
                initializer=tf.constant_initializer(0.1),
                shape=[HIDDEN_LAYER_SIZE]
            )
            result = tf.nn.relu(tf.matmul(result, weights) + biases)
            features = HIDDEN_LAYER_SIZE

            weights = tf.get_variable(
                "weights_out",
                initializer=tf.truncated_normal_initializer(stddev=0.1),
                shape=[features, DISCRIMINATOR_OUTPUT_SIZE],
            )
            biases = tf.get_variable(
                "biases_out",
                initializer=tf.constant_initializer(0.1),
                shape=[DISCRIMINATOR_OUTPUT_SIZE]
            )
    return tf.matmul(result, weights) + biases
```

Создадим также вектор сгенерированных примеров и прогоним все примеры через дискриминатор.

```
generated = generator(generator_input)
out_for_real = discriminator(input_data)
out_for_generated = discriminator(generated)
```

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

```
loss_real = tf.reduce_mean( # нахождение среднего по минибатчу
    tf.nn.sigmoid_cross_entropy_with_logits(
        labels=tf.ones_like(out_for_real), # массив единиц такого же размера
        logits=out_for_real # оценки дискриминатора
    )
)
```

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

```
loss_generated = tf.reduce_mean(
    tf.nn.sigmoid_cross_entropy_with_logits(
        labels=tf.zeros_like(out_for_generated),
        logits=out_for_generated
    )
)
```

Тогда функция потерь дискриминатора будет равна сумме потерь на реальных примера и на поддельных примерах. А функция потерь генератора – это кросс-энтропия между единицей (положительным ответом дискриминатора на поддельных примерах) и оценками дискриминатора:

```
generator_loss = tf.reduce_mean(
    tf.nn.sigmoid_cross_entropy_with_logits(
        labels=tf.ones_like(out_for_generated),
        logits=out_for_generated
    )
)
```

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.

```
with tf.variable_scope('generator_params') as scope:  
    generator_variables = [v for v in tf.global_variables() if v.name.startswith(scope.name)]  
  
with tf.variable_scope('discriminator_params') as scope:  
    discriminator_variables = [v for v in tf.global_variables() if v.name.startswith(scope.name)]  
  
generator_train = tf.train.AdamOptimizer().minimize(  
    generator_loss,  
    var_list=generator_variables,  
    name='train_generator'  
)  
  
discriminator_train = tf.train.AdamOptimizer().minimize(  
    discriminator_loss,  
    var_list=discriminator_variables,  
    name='train_discriminator'  
)
```

Посмотрим теперь графики обучения сетей в зависимости от шага обучения:

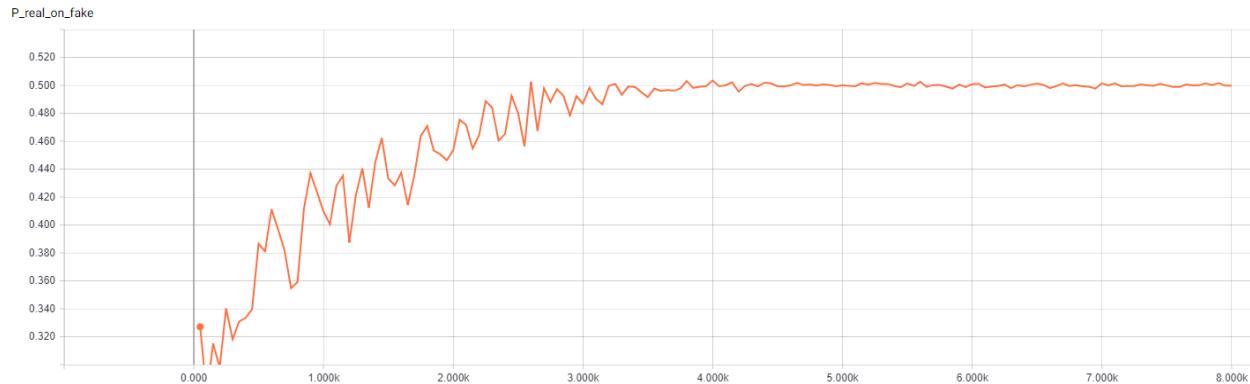


Рисунок 7. Вероятность классификации дискриминатором сгенерированного примера как реального

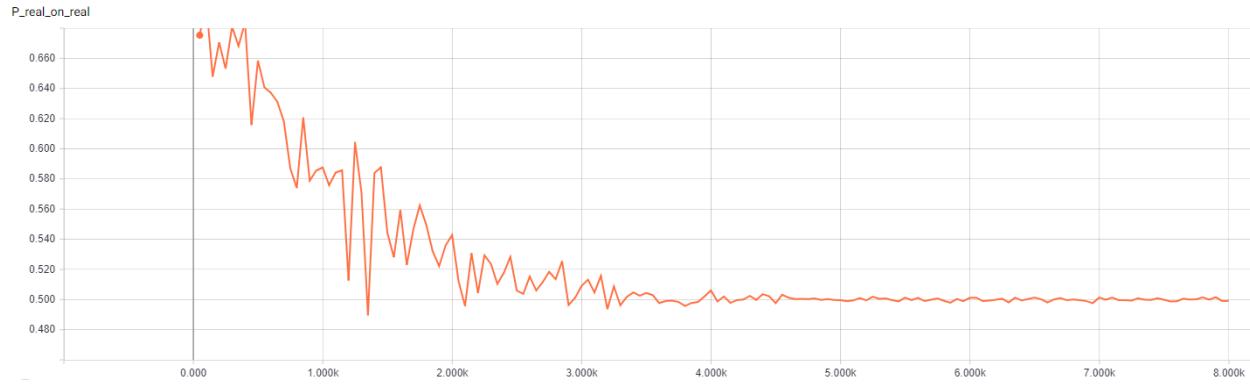


Рисунок 2. Вероятность классификации дискриминатором реального примера как реального

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

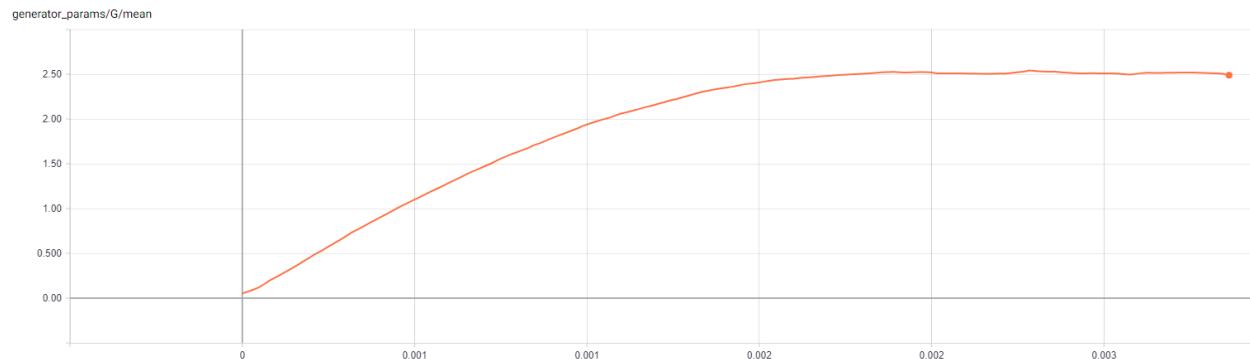


Рисунок 8. Мат. ожидание сгенерированных распределений

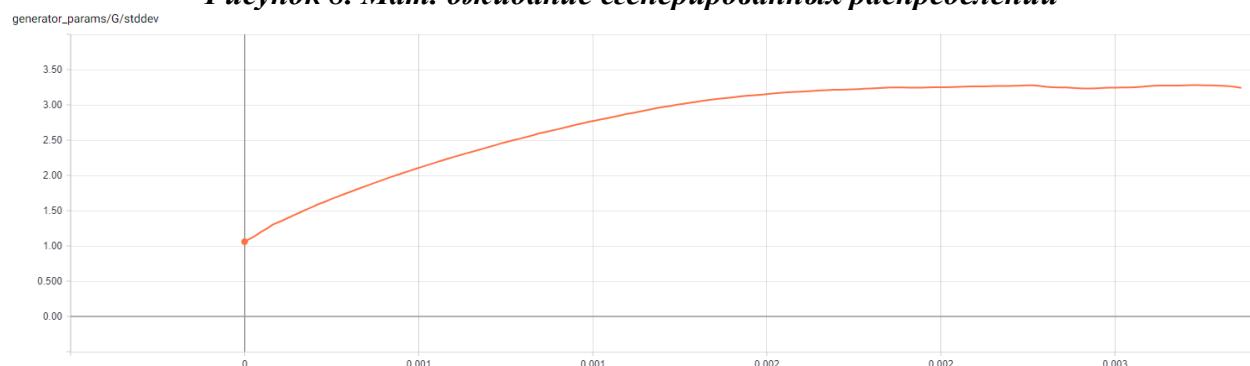


Рисунок 4. Дисперсия сгенерированных распределений

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 $p(x) = Mixture(N(\mu_1, \sigma_1), N(\mu_2, \sigma_2))$. Для этого надо изменить код генерации реальных примеров:

```
mean1 = 5.5
stddev1 = 3.25
mean2 = -3.2
stddev2 = 1.75
train_data_1 = np.random.normal(mean1, stddev1, size=(BATCH_SIZE // 2, 1))
train_data_2 = np.random.normal(mean2, stddev2, size=(BATCH_SIZE // 2, 1))
train_data = np.concatenate((train_data_1, train_data_2))
np.random.shuffle(train_data)
```

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

```
def generator(generator_input):
    result = generator_input
    features = 1
    with tf.variable_scope('generator_params', reuse=tf.AUTO_REUSE):
        for i in range(GEN_HIDDEN_LAYERS_NUMBER):
            weights = tf.get_variable(
                "gen_weights_%d" % i,
                initializer=tf.truncated_normal_initializer(stddev=0.1),
                shape=[features, GEN_HIDDEN_LAYER_SIZE]
            )
            biases = tf.get_variable(
                "gen_biases_%d" % i,
                initializer=tf.constant_initializer(0.1),
                shape=[GEN_HIDDEN_LAYER_SIZE]
            )
            result = tf.nn.relu(tf.matmul(result, weights) + biases)
            features = GEN_HIDDEN_LAYER_SIZE
```

```

weights = tf.get_variable(
    "gen_weights_out",
    initializer=tf.truncated_normal_initializer(stddev=0.1),
    shape=[features, 1],
)
biases = tf.get_variable(
    "gen_biases_out",
    initializer=tf.constant_initializer(0.1),
    shape=[1]
)
return tf.matmul(result, weights) + biases

```

Посмотрим теперь графики обучения сетей в зависимости от шага обучения:

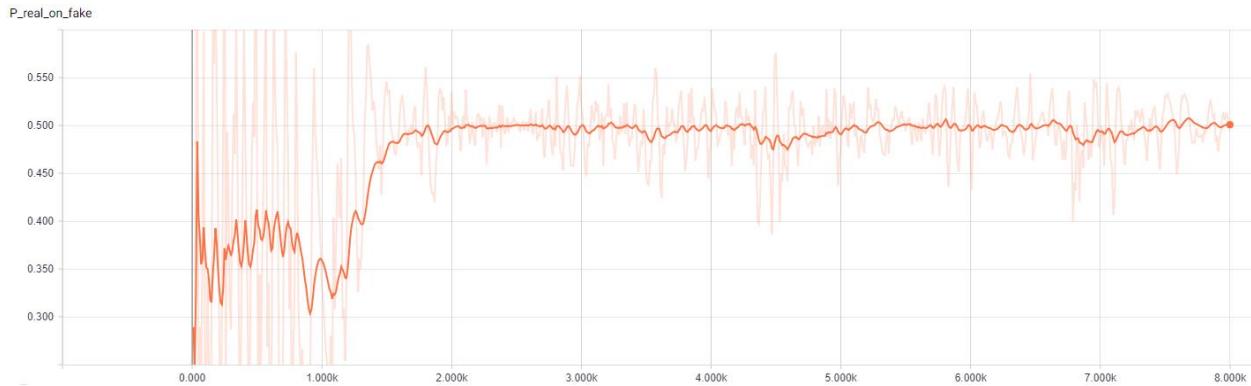


Рисунок 9. Вероятность классификации дискриминатором сгенерированного примера как реального

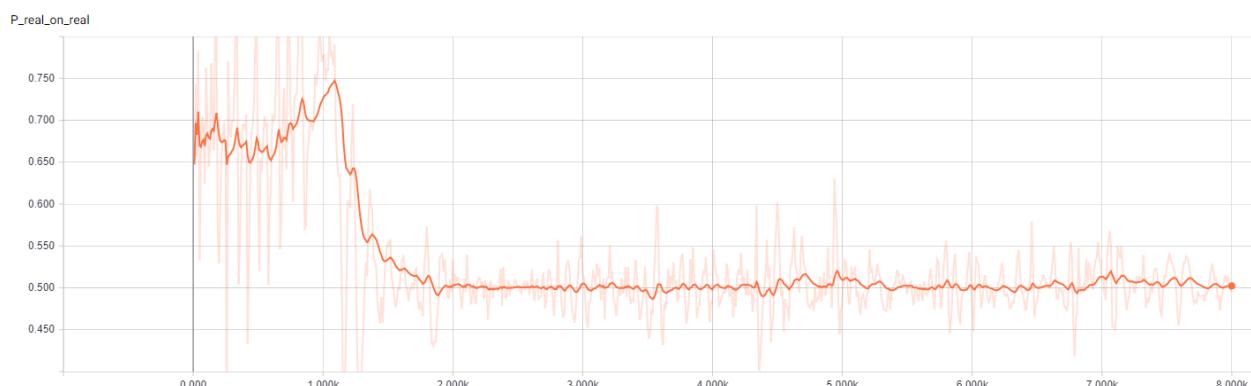


Рисунок 10. Вероятность классификации дискриминатором реального примера как реального

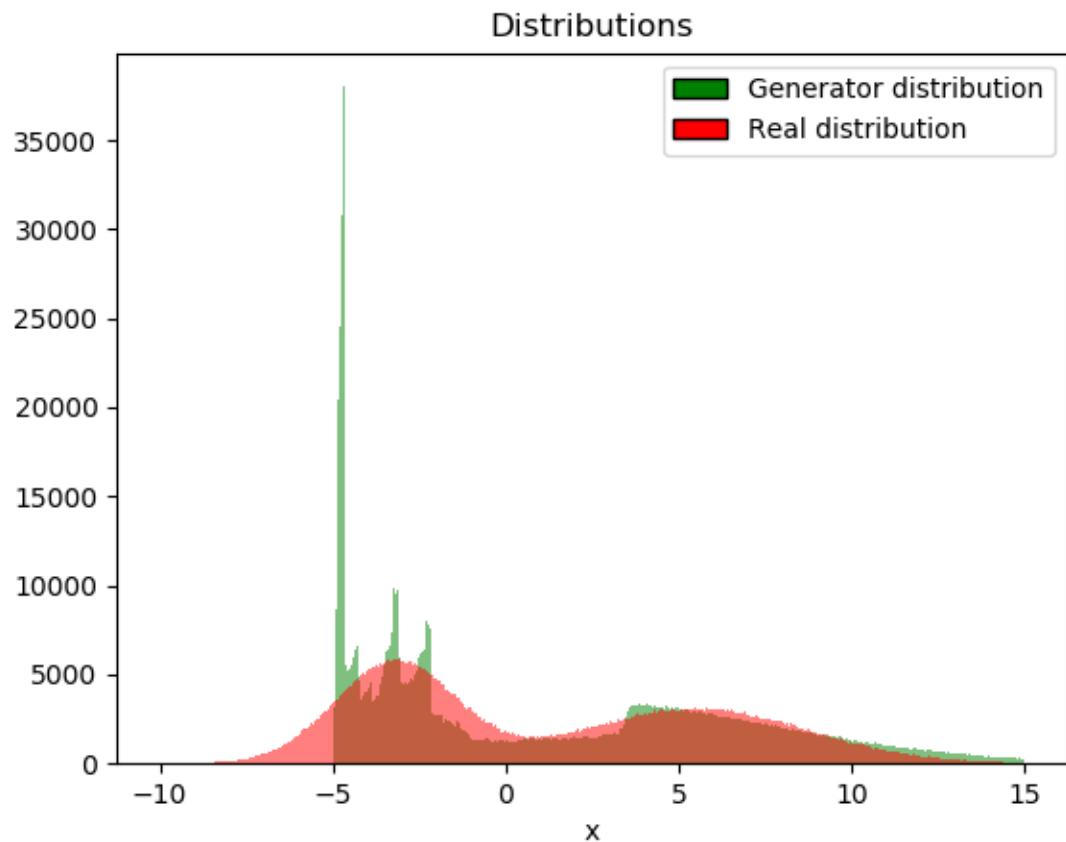


Рисунок 11. Распределения генератора и реальных данных

Видно, что распределение генератора хоть 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:

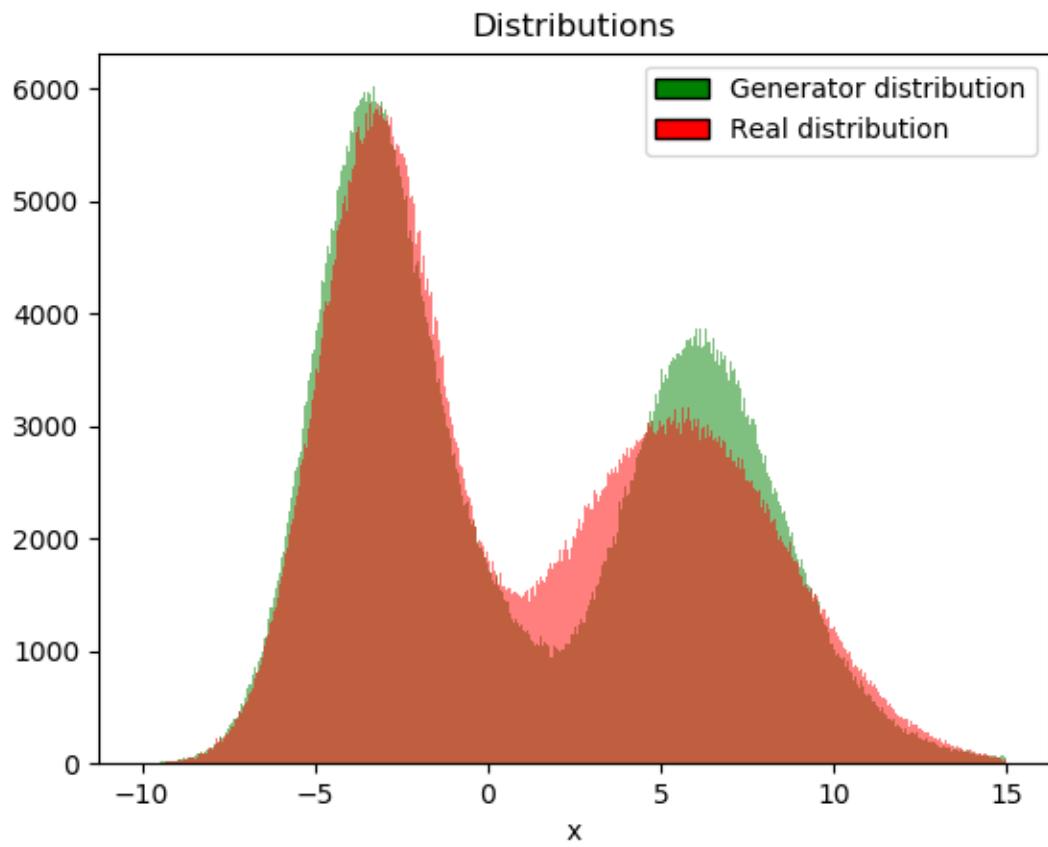


Рисунок 12. Распределения генератора и реальных данных

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).

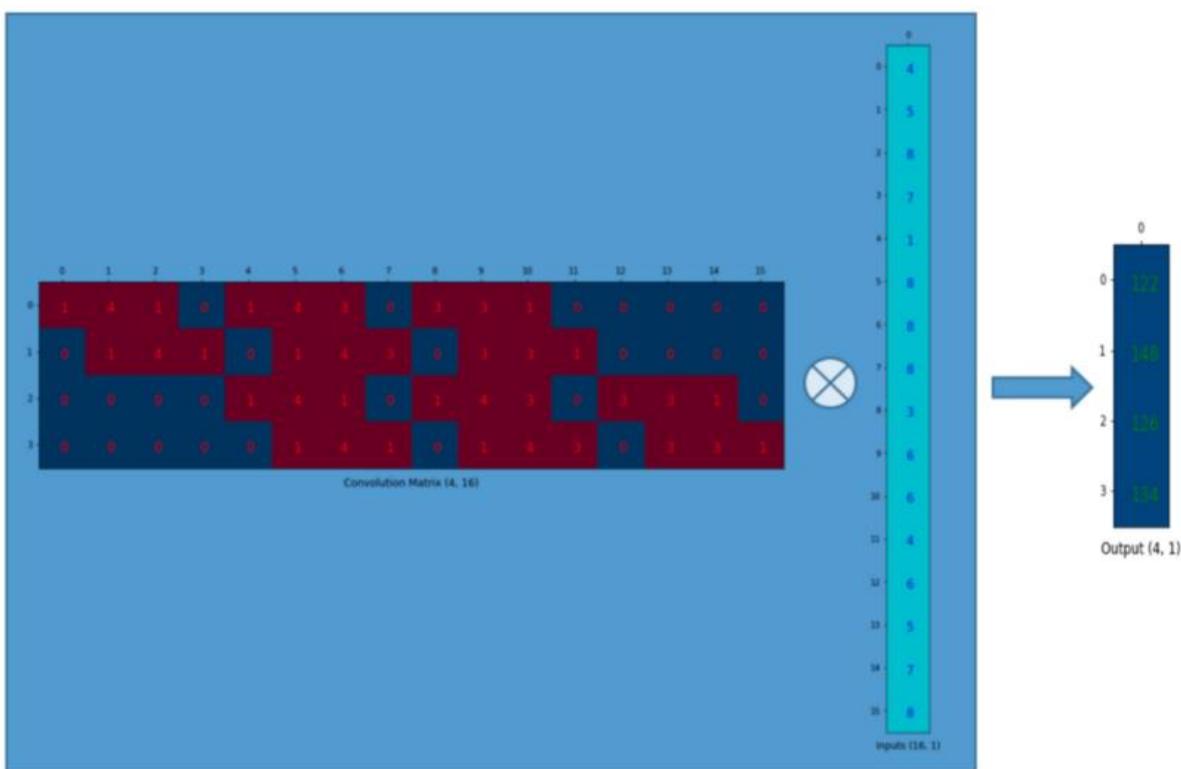


Рисунок 13. Обычная свёртка

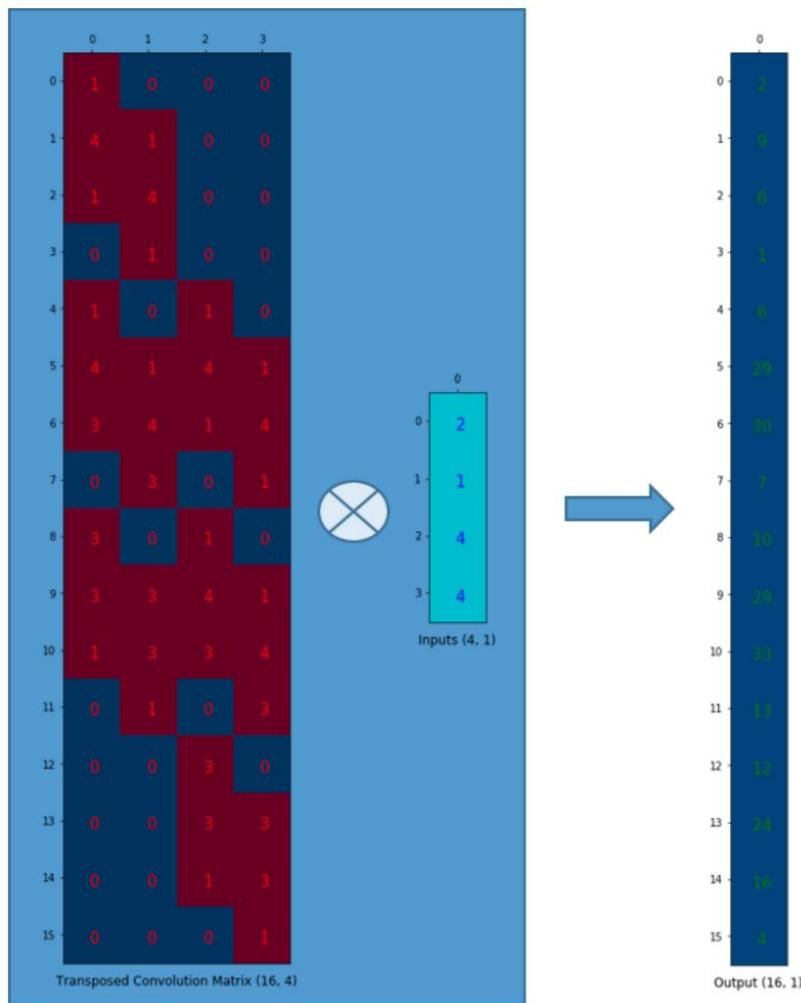


Рисунок 14. Транспонированная свёртка

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 связанным слоем и двумя слоями транспонированной свёртки, а дискриминатор будет обычной сетью с

двумя слоями свёртки и одним полно связанным слоем. Также добавим регуляризацию весов каждой нейронной сети, то есть введем штраф в функцию потери за норму всех обучаемых переменных.



Рисунок 15. 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	Three blurry, noisy grayscale images showing abstract patterns, likely generated by the generator at step 100.
1000	Ten blurry, noisy grayscale images showing abstract patterns, likely generated by the generator at step 1000.
2000	Two rows of ten sharp, clear wavy white lines on a black background, representing the final generated images at step 2000.



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

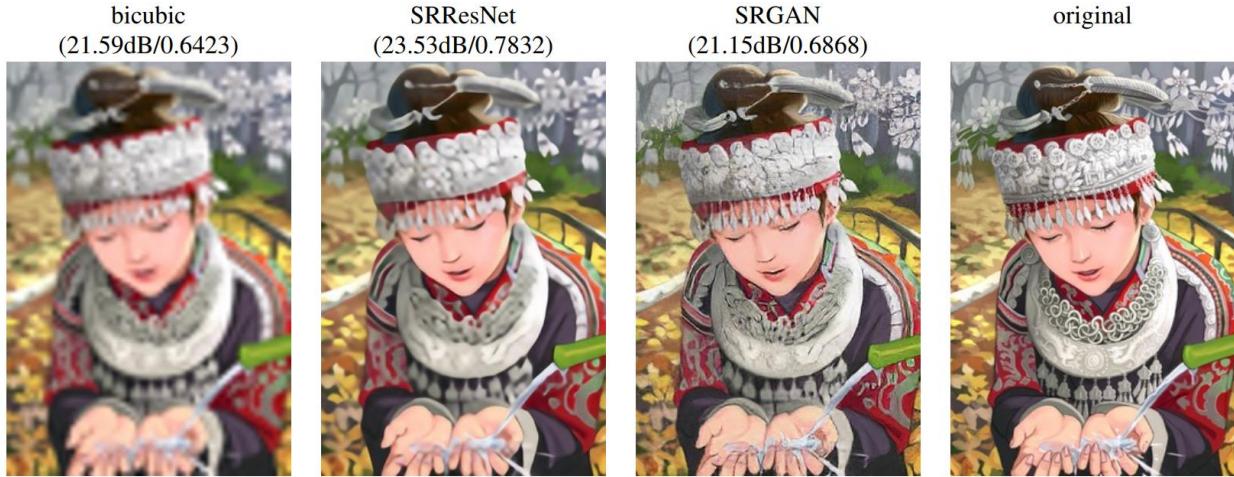
5. Супер-разрешения с помощью свёрточных генеративно-состязательных сетей

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 попиксельной 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 фотореалистичным, как определено в работе Ferwerda (http://cin.ufpe.br/~in1123/material/vor_hvei03_v20.pdf Фотореалистичное

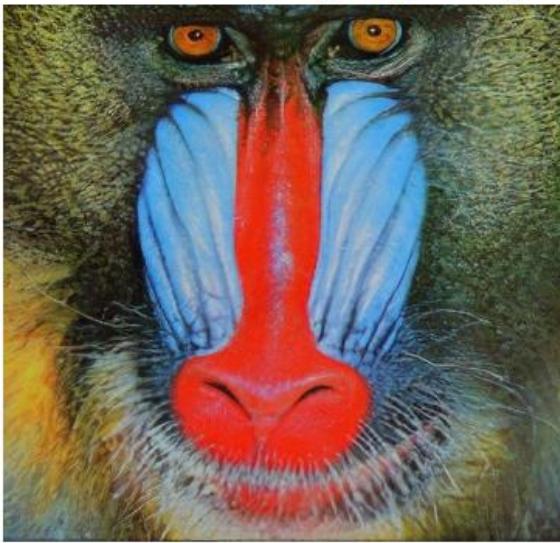
изображение производит тот же визуальный эффект как и реальная сцена).



*Рисунок 16. 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]*

В этой работе мы предлагаем генеративно состязательную сеть суперразрешения (SRGAN), в которой мы используем глубокую свёрточную сеть с residual связями (ResNet), с пропущенными соединениями и с целью оптимизации отличающейся от MSE. В отличие от предыдущих работ, мы 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.

$4 \times$ SRGAN (proposed)



original

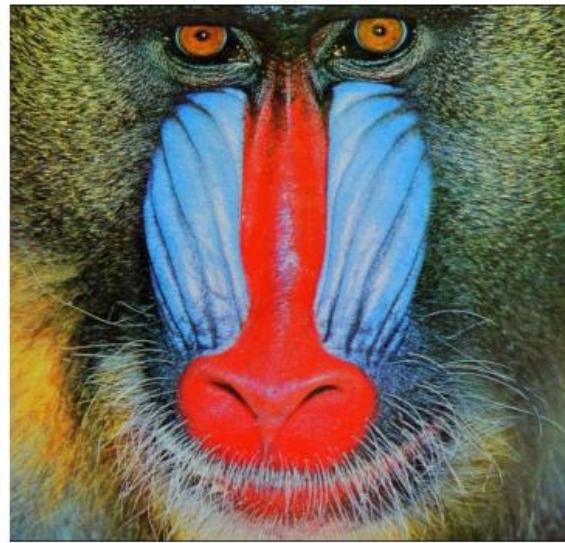


Рисунок 17. Суперразрешение изображения (слева) почти неотличимо от оригинального (справа). [4х масстабирование]

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_{θ_G} , зависящую от параметров θ_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_{θ_D} , которую мы оптимизируем поочередно наряду с G_{θ_G} для решения состязательной мин-макс проблемы:

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

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 3×3 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 3×3 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.

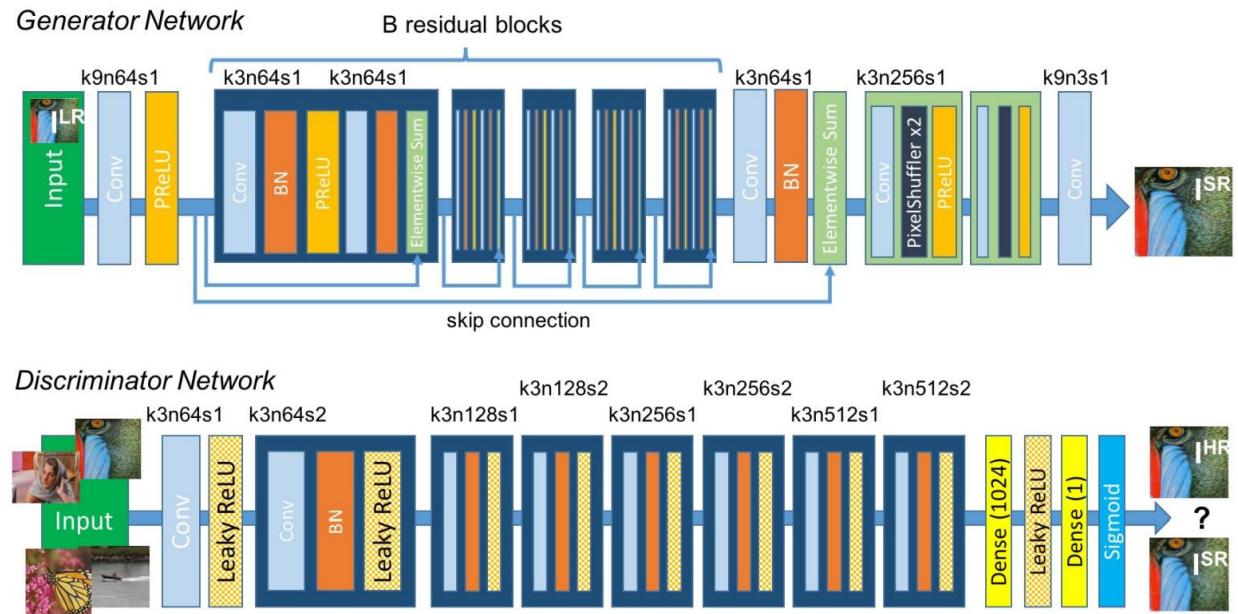


Рисунок 18. Архитектура Генерирующей и Дискриминирующей сети с соответствующими размерами ядер (k), количествами карт особенностей (n) и шагами (s), которые показаны для каждого свёрточного слоя.

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

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/lcue6vlrd01ljkdtdkhnfvk7vtjhetog>). 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 96×96 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 ≈ 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 дискриминатора, 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.

Тестирование средней оценки мнения (Mean opinion score = MOS)

Мы выполнили MOS тест для количественной оценки способности различных подходов восстанавливать визуальное восприятие изображений. В частности, мы опросили 26 оценщиков присвоить интегральный балл от 1 (плохое качество) до 5 (отличное качество) для суперразрешённых изображений. Оценщики оценили 12 версий каждого изображения на датасетах Set5, Set14 и BSD100: метод ближайшего соседа, бикубическая интерполяция, SRCNN, SelfExSR, DRCN, ESPCN, SRResNet-MSE, SRResNet-VGG22* (*не оценено на BSD100), SRGAN-MSR*, SRGAN-VGG22*, SRGAN-VGG54 и оригинальное HR изображение. Каждый оценщик оценил 1128 изображений, которые были представлены в случайном порядке. Оценщики были откалиброваны на NN (оценка 1) и HR (оценка 5) версиях 20 изображений из датасета BSD300. В пробном исследовании мы оценили процедуру калибровки и надежность повторного тестирования 26 оценщиков на подмножестве из 10 изображений из BSD100, добавив изображения из различных методов дважды в большой тестовый набор. Мы нашли хорошую надежность и никаких существенных различий между оценками идентичных

изображений. Оценщики очень последовательно оценивали NN-интерполированные тестовые изображения как 1 и оригинальные HR изображения как 5 (см. Рисунок 4).

Экспериментальные результаты проведенных MOS-испытаний описаны в Таблице 1, Таблице 2 и на Рисунке 5.

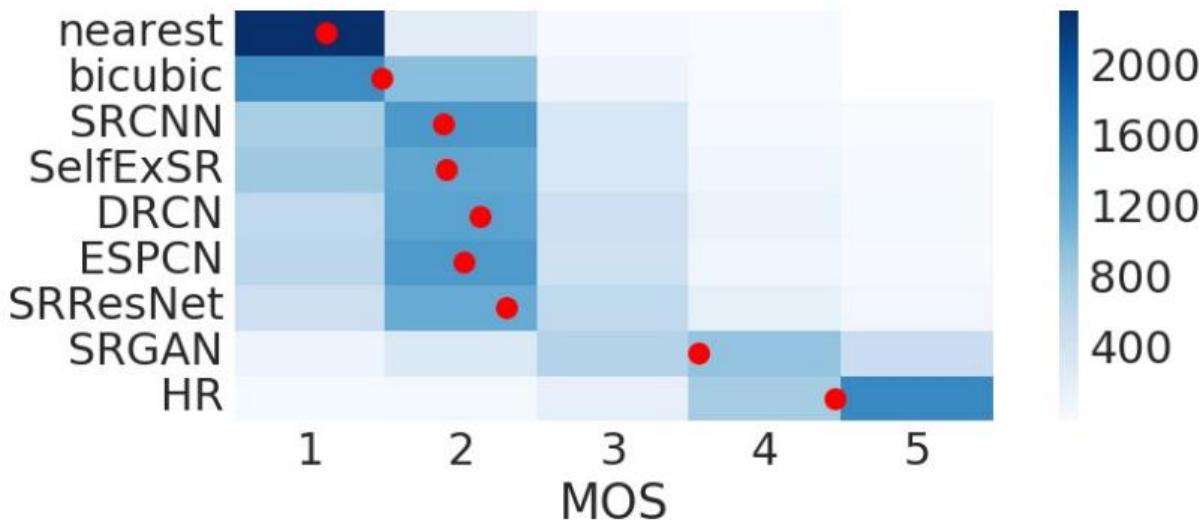


Рисунок 19. Распределение MOS баллов закодированное в цвете на BSD100. Для каждого метода 2600 образцов было оценено (100 изображений x 26 оценщиков). Среднее значение отображается красным маркером, где ячейки центрированы около значения i. [4x масштабирование]

	SRResNet-		SRGAN-		
Set5	MSE	VGG22	MSE	VGG22	VGG54
PSNR	32.05	30.51	30.64	29.84	29.40
SSIM	0.9019	0.8803	0.8701	0.8468	0.8472
MOS	3.37	3.46	3.77	3.78	3.58
Set14					
PSNR	28.49	27.19	26.92	26.44	26.02
SSIM	0.8184	0.7807	0.7611	0.7518	0.7397
MOS	2.98	3.15*	3.43	3.57	3.72*

Таблица 1. Результаты различных функций потерь для SRResNet и состязательных сетей на Set5 и Set14 датасетах. MOS значительно выше ($p<0.05$) чем с другими потерями в этой категории [4x масштабирование]

Set5	nearest	bicubic	SRCNN	SelfExSR	DRCN	ESPCN	SRResNet	SRGAN	HR
PSNR	26.26	28.43	30.07	30.33	31.52	30.76	32.05	29.40	∞
SSIM	0.7552	0.8211	0.8627	0.872	0.8938	0.8784	0.9019	0.8472	1
MOS	1.28	1.97	2.57	2.65	3.26	2.89	3.37	3.58	4.32
Set14									
PSNR	24.64	25.99	27.18	27.45	28.02	27.66	28.49	26.02	∞
SSIM	0.7100	0.7486	0.7861	0.7972	0.8074	0.8004	0.8184	0.7397	1
MOS	1.20	1.80	2.26	2.34	2.84	2.52	2.98	3.72	4.32
BSD100									
PSNR	25.02	25.94	26.68	26.83	27.21	27.02	27.58	25.16	∞
SSIM	0.6606	0.6935	0.7291	0.7387	0.7493	0.7442	0.7620	0.6688	1
MOS	1.11	1.47	1.87	1.89	2.12	2.01	2.29	3.56	4.46

Таблица 2. Сравнение NN, бикубическая, SRCNN, SelfExSR, DRCN, ESPCN, SRResNet, SRGAN-VGG54 и оригинальное HR изображение на оценочных данных. [4x масштабирование]

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 фокусированным 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 $l_{VGG/2.2}^{SR}$. Количественные 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 не можем определить существенно лучшую функцию потерь для SRResNet или SRGAN в отношении оценки MOS на Set5. Однако SRGAN-VGG54 значительно превзошел другие SRGAN и SRResNet варианты на Set14 с точки зрения MOS. 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).

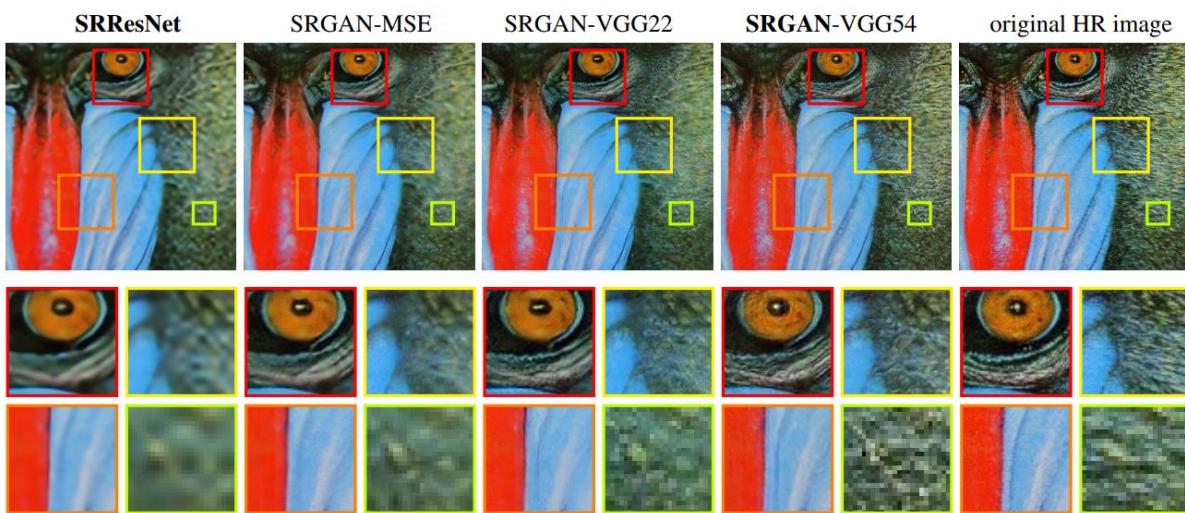


Рисунок 20. SRResNet, SRGAN-MSE, SRGAN-VGG22, SRGAN-VGG54 результаты восстановления и соответствующее HR изображение [4x масштабирование]

Результаты финальных сетей

Мы сравниваем производительность **SRResNet** и **SRGAN** с методом ближайшего соседа, бикубической интерполяцией, и четырьмя современными методами. Количественные результаты суммированы в Таблице 2 и

подтверждают, что **SRResNet** (с точки зрения PSNR/SSIM) устанавливает новый уровень качества на трех эталонных датасетах.

Далее мы получили рейтинги MOS для **SRGAN** и для всех эталонных методов на BSD100. Примеры суперразрешенных изображений с помощью **SRResNet** и **SRGAN** приведены на Рисунке 6. Результаты, показанные в Таблице 2 подтверждают, что SRGAN превосходит все эталонные методы с большим отрывом и устанавливает новый уровень качества для фотореалистичных изображений суперразрешения. Все различия MOS (см. Таблицу 2) очень значительны на **BSD100**, за исключением SRCNN vs. SelfExSR. Распределение всех собранных MOS оценок суммированы на Рисунке 4.



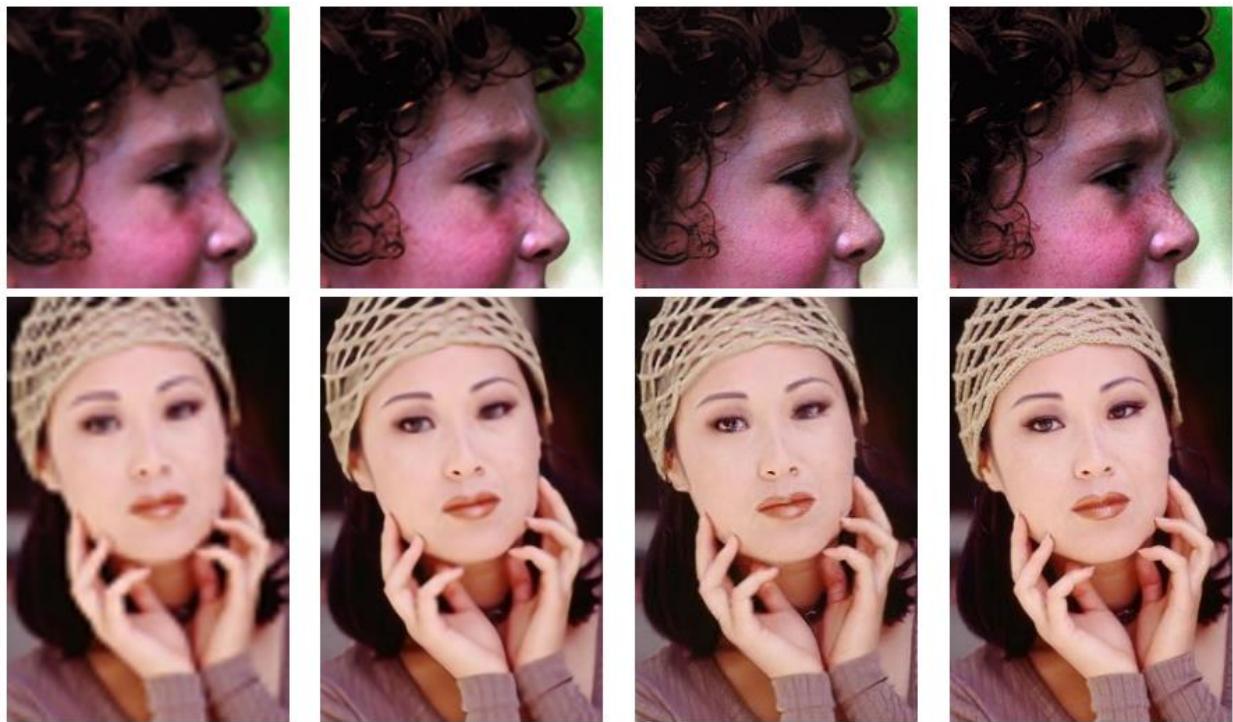


Рисунок 21. Результаты из датасета Set5 используя бикубическую интерполяцию, SRResNet и SRGAN. [4x масштабирование]

LR изображение HR изображение SR изображение



Результаты обученной модели

5.4 Возможные улучшения

Подтвердили превосходное визуальное восприятие **SRGAN** используя MOS тестирование. Также показали, что стандартные количественные метрики как PSNR и SSIM не в состоянии оценить качество изображения по отношению к зрительной системе человека. Целью этой работы было качество визуального восприятия суперразрешенных изображений, а не вычислительная эффективность. Представленная модель, в отличие от ESPCN в работе Shi et al. (<https://arxiv.org/pdf/1609.05158.pdf>), не оптимизирована для суперразрешения видео в режиме реального времени. Однако предварительные эксперименты с сетевой архитектурой предполагают, что более мелкие сети могут предоставить очень эффективные альтернативы при небольшом уменьшении качества результата. В отличие от Dong et al. (SRCNN), мы сочли полезным более глубокие сетевые архитектуры. Мы предполагаем, что дизайн ResNet оказывает существенное влияние на результаты более глубоких сетей. Мы обнаружили, что даже более глубокие сети ($B > 16$) могут еще улучшить результаты **SRResNet**, однако за счет увеличения времени обучения и тестирования. Далее мы обнаружили, что варианты SRGAN для более глубоких сетей все труднее обучать из-за появления высокочастотных артефактов.

Особое значение при стремлении к фотореалистичным результатам проблемы суперразрешения имеет выбор потери содержания, как показано на Рисунке 5. В этой работе мы нашли, что $l_{VGG}^{SR}/5.4$ дает наиболее убедительный результат для восприятия, атрибутом этого является потенция глубоких слоёв сети для предоставления признаков высокой абстракции вне пространства пикселей. Мы предполагаем, что карты особенностей этих глубоких слоёв

фокусируются исключительно на содержании, пока состязательные потери сфокусированы на текстурных деталях, которые являются основным отличием между суперразрешенными изображениями без состязательной потери и фотореалистичными изображениями. Также отметим, что идеальная функции потери зависит от применения. Например, подходы, которые галлюцинируют более мелкие детали, могут быть менее подходящими для медицинского применения или наблюдения. Убедительная в восприятии реконструкция текста или структурирование сцен является сложной задачей и частью будущих работ. Развитие функций потери содержания, которые описывают пространственное содержание изображения, но более инвариантны к изменениям в пиксельном пространстве, дополнительно улучшат результаты суперразрешения фотореалистичного изображения.

5.5 Выводы

В этой работе описан новый стандарт для фотореалистичного 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 MOS тестирование, описанное в данной работе, которое показывает явные преимущества SRGAN подхода в создании фотореалистичных изображений.

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 одно изображение более реалистичным, чем другое", в отличие "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).

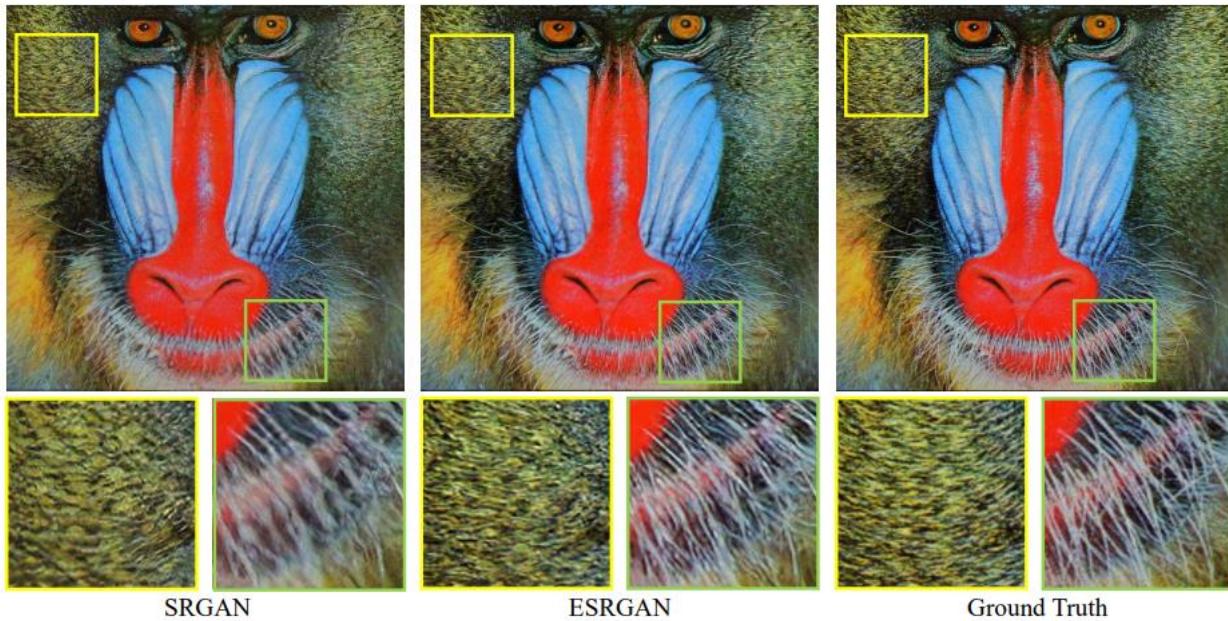


Рисунок 1. Результаты супер-разрешения x4 для SRGAN, предложенный ESRGAN и реальное изображения. ESRGAN превосходит SRGAN в резкости и деталях.

В PIRM-SR соревновании первым SR задачей является оценивание эффективности с учетом качества восприятия на основе работы о компромиссе между восприятием и искажением, то есть они противоречат друг другу. О качестве восприятия судят по оценке Ma (<https://arxiv.org/pdf/1612.05890.pdf>) и NIQE (<https://ieeexplore.ieee.org/document/6353522>), то есть индекс восприятия $= 1/2((10 - Ma) + NIQE)$. Самый низкий индекс восприятия представляет лучшее лучшее качество восприятия.

Как показано на Рисунке 2, кривая восприятия-искажения поделена на 3 региона, определенные пороговыми значениями для среднеквадратичной

ошибки (Root-Mean-Squared Error = RMSE), и алгоритм, который достигает наименьшего индекса восприятия в каждой области становится региональным чемпионом. В основном сфокусируемся на регионе 3, поскольку стремимся поднять качество восприятия на новый уровень. Благодаря вышеупомянутым улучшениям и некоторым другим корректировкам, как описано в следующих главах, предложенный ESRGAN занимает первое место в PIRM-SR соревнование (регион 3) с лучшим индексом восприятия.

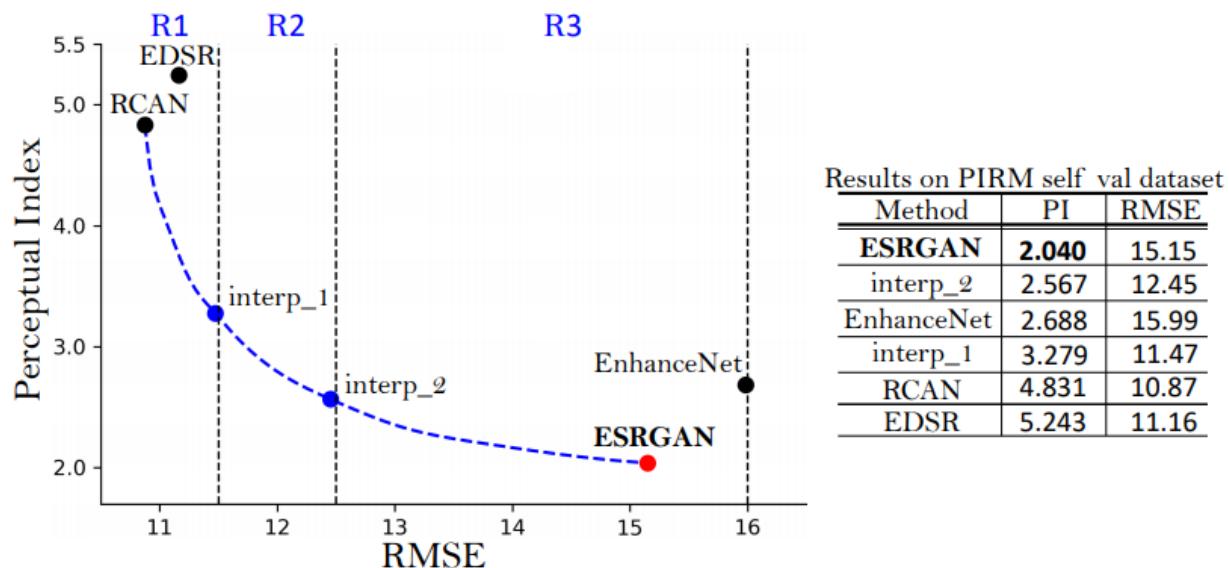


Рисунок 2. Плоскость восприятия-искажения на наборе данных самопроверки PIRM. Показаны основные модели: EDSR, RCAN, EnhanceNet и предложенный ESRGAN. Синие точки получаются интерполяцией изображения.

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. Эту 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.

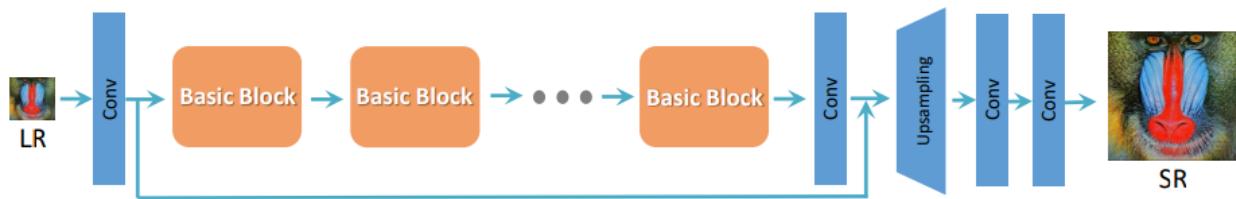


Рисунок 3. Используется стандартная архитектура SRResNet, в которой большая часть вычислений делается в пространстве особенностей LR. Можно изменить дизайн Basic Blocks (residual block, dense block, RRDB) для лучшей производительности.

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 4.

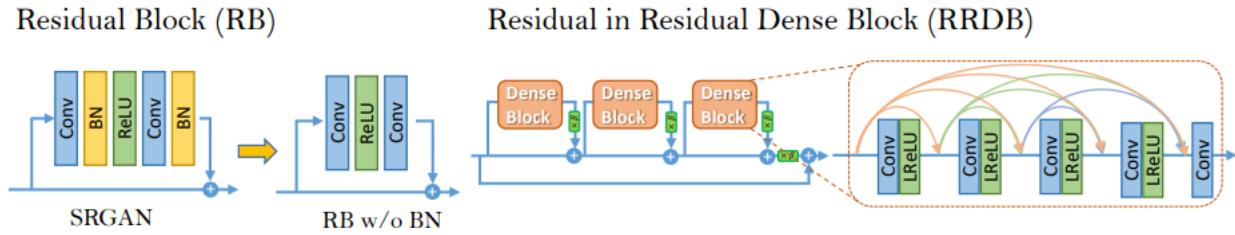


Рисунок 4. Левый: Удалены BN слои из residual блока в SRGAN. Правый: RRDB блок использующийся в новой глубокой модели и β как residual масштабирующий параметр.

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 4. 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 4, 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 5.

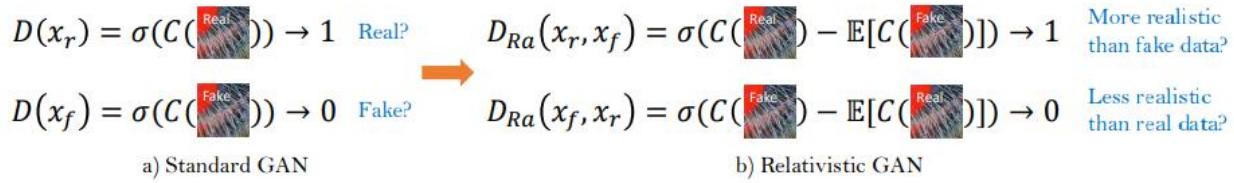


Рисунок 5. Разница между стандартным дискриминатором и релятивистским дискриминатором.

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 σ – 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 идеи 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 6. 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 λ, η – 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 для SR - MINC потеря. 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.

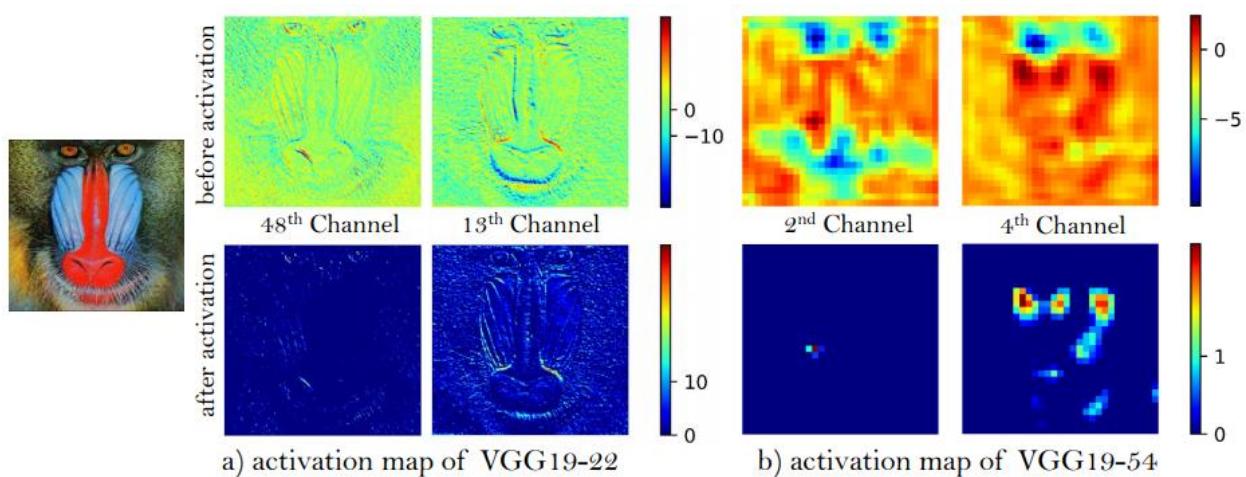


Рисунок 6. Репрезентативные карты особенностей перед и после активации для изображения «baboon». По мере углубления сети большинство особенностей после активации становится неактивными, в то время как особенности перед активацией содержат больше информации.

Сетевая интерполяция

Чтобы устраниить 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:

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

где θ_G^{INTERP} , θ_G^{PSNR} и θ_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 α 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

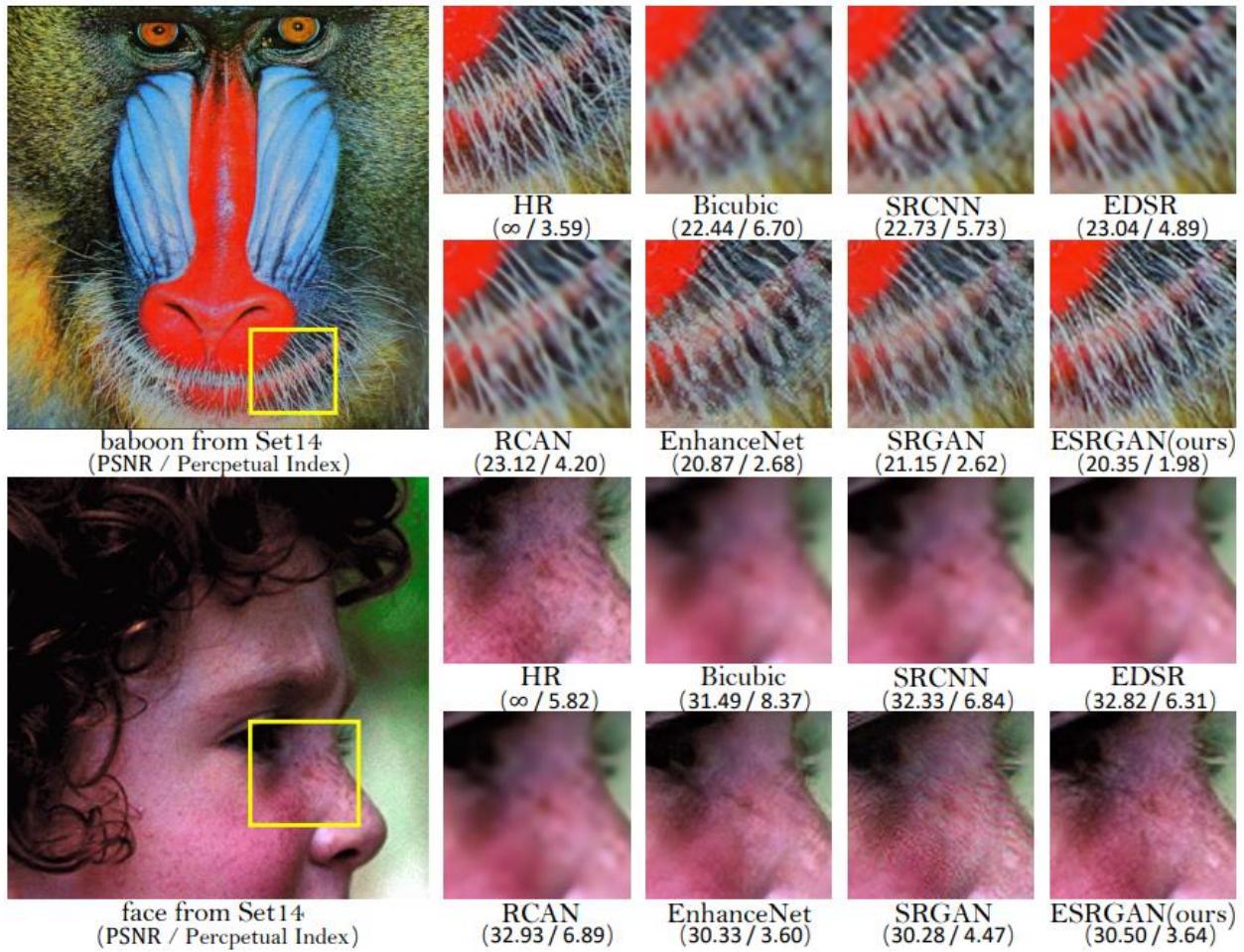
Для обучения использовался в основном датасет DIV2K, в котором изображения высокого разрешения (2K+) для задач восстановления изображений. Помимо учебного набора из DIV2K, который содержит 800 изображений, также использовались другие датасеты со сложными и разнообразными текстурами для обучения. Для 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 8.

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 и на наборе данных самопроверки PIRM, предоставленном от PIRM-SR соревнования.

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 7. PSNR (считается по каналу яркости в цветовом представлении YCbCr) и индекс восприятия, используемый в PIRM-SR соревновании также представлены для справки.

Как можно заметить на Рисунке 7 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.



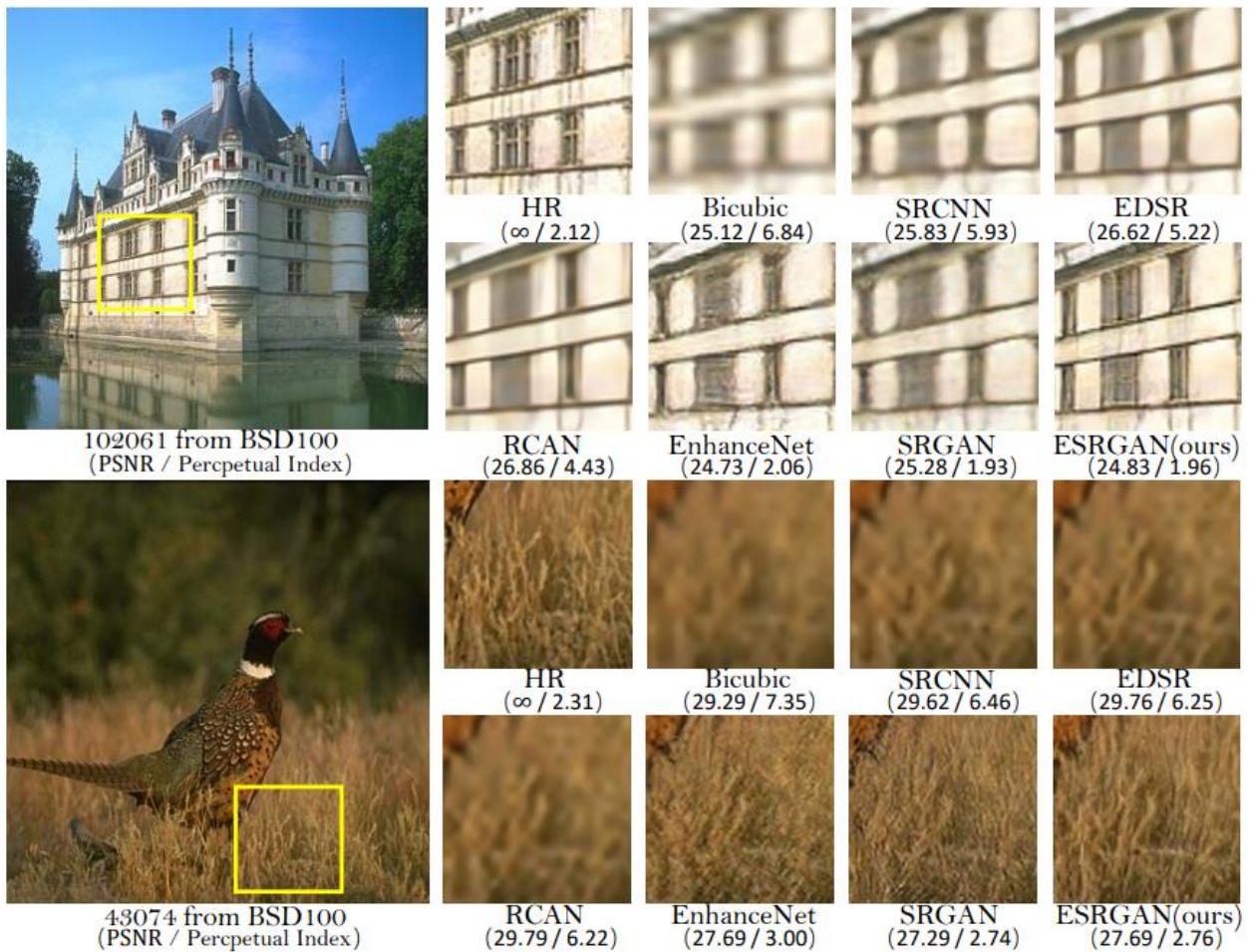


Рисунок 7. Качественные результаты ESRGAN. ESRGAN получает более натуральные текстуры, например, шерсть животных, строительные структуры и текстура травы, а также меньше нежелательных артефактов, например, артефакты на лице от SRGAN.

Исследование аблляции

Чтобы изучить влияние каждой компоненты в предложенном ESRGAN, будем постепенно модифицировать базовую модель SRGAN и сравнивать их разницу. Общее визуальное сравнение показано на Рисунке 9. 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 9 (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 8.

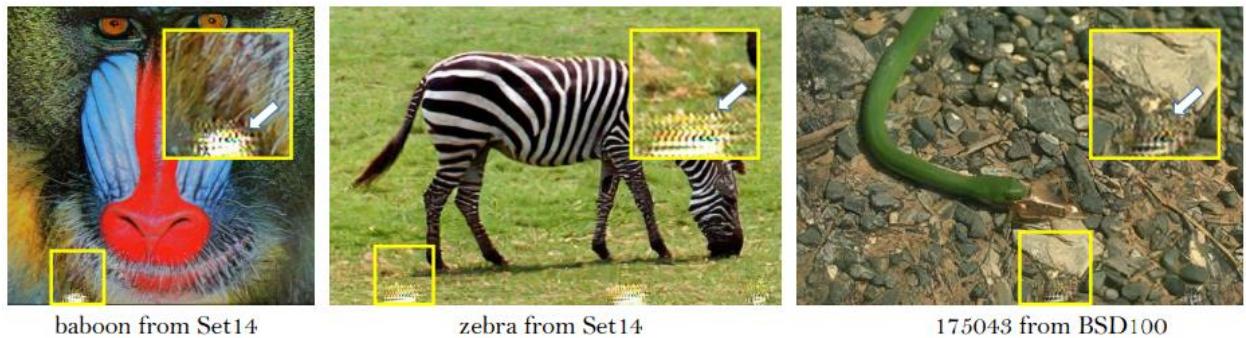
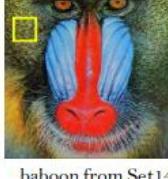
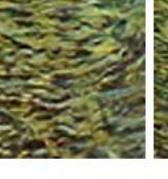
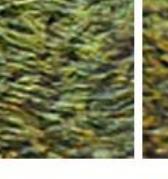
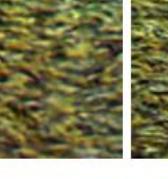
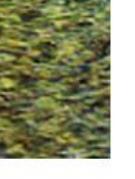
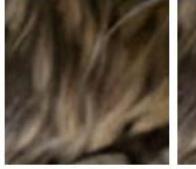
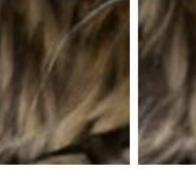


Рисунок 8. Пример BN артефактов в GAN модели.

1 st	2 nd	3 rd	4 th	5 th	6 th	7 th
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						

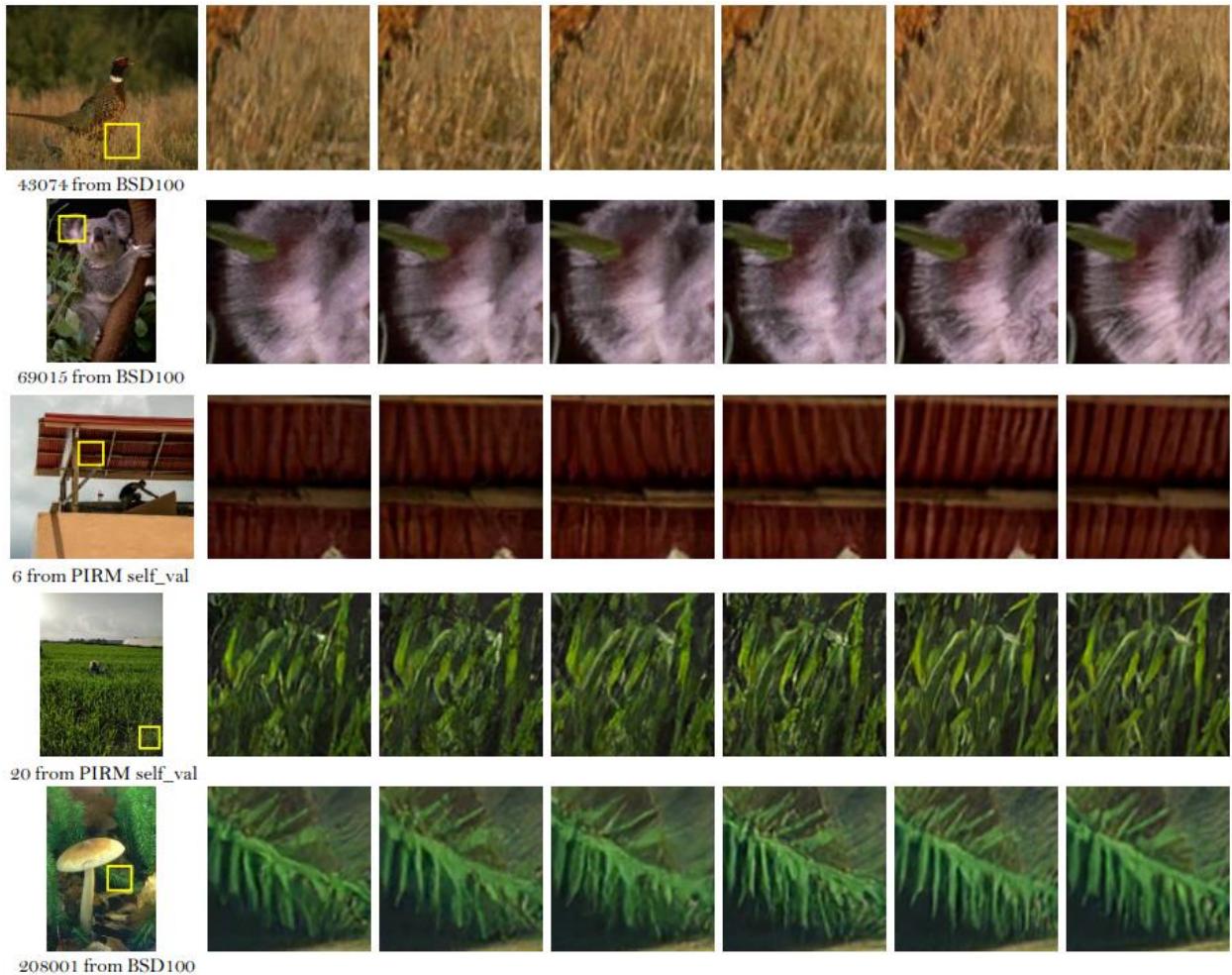


Рисунок 9. Общее визуальное сравнение для демонстрации эффектов каждой компоненты в ESRGAN. Каждый столбец отображает модель с конфигурацией, описанной наверху. Красный знак указывает на ключевое улучшение по сравнению с предыдущей моделью.

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 10а 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 10b (см. перо птицы) и на Рисунке 9 (см. 3-й и 4-й столбцы), поскольку плотность особенностей перед активацией предлагает более сильный контроль, чем его может обеспечить редкие активации.

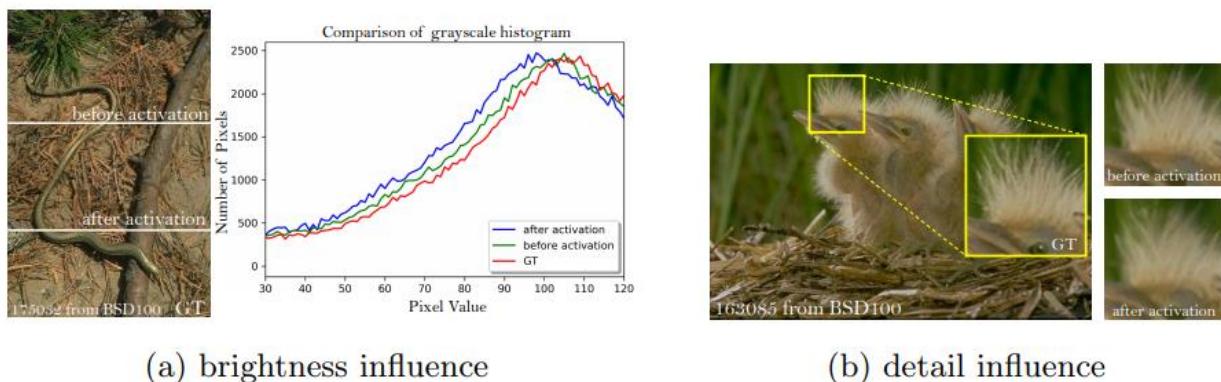


Рисунок 10. Сравнение между взятием особенностей перед активацией или после активации.

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 9 генерированные изображения более четкие и с более богатыми текстурами, чем слева (см. baboon, изображение 39 и изображение 43074).

Более глубокая 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 9, 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 9.

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 11, 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 11.



Рисунок 11. 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 дискrimинатора, 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. Заключение

8. 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]
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]

10. 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.