Федеральное государственное автономное образовательное учреждение высшего образования
Национальный исследовательский университет
«Высшая школа экономики»

Факультет компьютерных наук

Долматов Кирилл Игоревич

РАЗРАБОТКА СЕРВИСА ДЛЯ УСЛОВНОЙ ГЕНЕРАЦИИ РЕАЛИСТИЧНЫХ ИЗОБРАЖЕНИЙ С ПОМОЩЬЮ НЕЙРОСЕТЕВЫХ МОДЕЛЕЙ

Выпускная квалификационная работа

студента образовательной программы магистратуры «Машинное обучение и высоконагруженные системы» по направлению подготовки *01.04.02 Прикладная математика и информатика*

Рецензент
ученая степень, ученое звание,
должность

Руководитель
ученая степень, ученое
звание, должность

Айбек Аланов
И.О. Фамилия

Москва, 2023 год

Аннотация

Условная генерация изображений может быть использована для задач создания уникального визуального контента, нетривиальной трансформации и аугментации изображений. Отличительной особенностью условной генерации является наличие условия, которое должно быть учтено при создании изображения, при этом множество условий является конечным и, как правило, относительно небольшим.

В данной работе выполнен обзор современных нейросетевых архитектур используемых для условной генерации изображений, а также проведены эксперименты на различных наборах данных, связанные с воспроизведением, сравнением и улучшением выбранных методов.

Результатом работы является созданный программный код, позволяющий конечному пользователю решать задачу условной генерации изображений посредством созданного веб-сервиса.

Abstract

Conditional image generation can be used for the tasks of creating unique visual content, non-trivial transformation and image augmentation. A distinctive feature of conditional generation is the presence of a condition that must be taken into account when creating an image, while the set of conditions is finite and, as a rule, relatively small.

This paper provides an overview of modern neural network architectures used for conditional image generation, as well as experiments on various data sets related to the reproduction, comparison and improvement of selected methods.

The result of the work is the created program that allows the end user to solve the problem of conditional image generation through the created web service.

Список ключевых слов

Условная генерация изображений, генеративные состязательные сети, conditional image generation, GAN, DCGAN, StyleGAN

Оглавление

Введение

Методы генерации изображений начали активно развиваться только в течение последних восьми лет, с тех пор как впервые была предложена принципиально новая архитектура - генеративно состязательная нейронная сеть.

Создание данной архитектуры позволило решать задачи, которые ранее считались недоступными для алгоритмов, а полученные результаты вполне сопоставимы с реальными изображениями.

Сегодня генеративные состязательные сети используются для решения различных задач, таких как генерация изображений, редактирование изображений, перенос стиля изображения, детектирование объектов и сегментация изображений.

Задача условной генерации изображений в отличие от обычной генерации изображений предполагает наличие условия, которое должно быть учтено при создании изображения. При этом множество условий является конечным и, как правило, относительно небольшим. В качестве условия может выступать метка класса изображения, которое мы хотим сгенерировать, а может выступать и некоторое изображение. Использование заданного условия позволяет в некоторой степени контролировать процесс генерации изображений и получать конечный результат, более соответствующий решаемой задаче.

С помощью методов условной генерация можно создавать уникальные и реалистичные изображения, что позволяет существенно экономить на расходах, связанных с созданием визуального контента. Кроме того, данные методы позволяют бороться с несбалансированностью выборок, путем генерации объектов минорных классов, что в итоге повышает качество прогнозирования конечных моделей.

В данной работе подробно рассматриваются современные методы условной генерации изображений, на примере нескольких наборов данных, но полученные результаты могут быть распространены и на другие наборы данных. Таким образом, объектом исследования является изображение, полученное по результатам работы модели, а предметом исследования является сам процесс условной генерации данного изображения.

Цель работы - создание комплексного прикладного программного решения, то есть продукта, который может быть использован конечным потребителем для

решения задачи условной генерации изображения. Поставленная цель включает выполнение следующих шагов:

- Выполнить обзор существующих методов условной генерации изображений
- Привести описание классических и доменных наборов данных
- Провести сравнительные эксперименты различных нейросетевых архитектур на имеющихся наборах данных и отобрать лучшие
- Реализовать условную генерации изображений в виде полноценного веб-сервиса
- Подтвердить функциональность и корректность работы созданного веб-сервиса

Глава 1 Теоретическая часть

1.1 Условные обозначения

1.2 Постановка задачи

Условная генерация изображения включает в себя использование набора входных изображений и конечного набора параметров для генерации выходных изображений с определенными характеристиками.

Наиболее распространенными используемыми алгоритмами являются генеративные состязательные сети (GAN) и сверточные нейронные сети (CNN). GAN генерируют изображения, обучая две нейронные сети — генератор и дискриминатор — совместной работе, в то время как CNN используют слои свертки для извлечения признаков из входных изображений. Также в последнее время большую популярность набирают диффузионные модели.

Как только соответствующий алгоритм выбран, входные изображения и параметры должны быть введены в алгоритм для начала процесса генерации. Затем алгоритм создаст выходное изображение, которое было сгенерировано условно в соответствии с предоставленными критериями.

Полученное изображение можно использовать для различных целей, таких как создание рисунков, распознавание объектов или даже манипулирование существующими изображениями. Кроме того, генерация условных изображений может быть использована для повышения точности моделей распознавания изображений за счет предоставления большего и более разнообразного набора данных.

Таким образом, благодаря достижениям в области алгоритмов глубокого обучения, генерация изображений является востребованным направлением как для научных исследований, так и практического применения с самых различных сферах. Возможность генерировать изображения с заданным условием или на основе определенных критериев позволяет управлять процессом генерации и решать более сложные прикладные задачи.

1.3 Обзор методов условной генерации изображений

1.3.1 GAN

Генеративно состязательные сети (англ. Generative Adversarial Net, GAN) – алгоритм машинного обучения, принадлежащий к классу порождающих моделей. В основе данного алгоритма лежит комбинация двух нейронных сетей: генеративная модель, строящая приближение распределения исходных входных данных, и дискриминативная модель, которая оценивает вероятность, что объект был получен из исходных данных (обучающей выборки), а не сгенерирован генеративной моделью. Генератор создает данные из случайного шума, в то время как дискриминатор оценивает сгенерированные данные и определяет, являются ли они реальными или сгенерированными. Этот процесс повторяется до тех пор, пока дискриминатор больше не сможет отличать реальные данные от сгенерированных. Впервые данная архитектура была представлена Иэном Гудфеллоу в 2014 году.

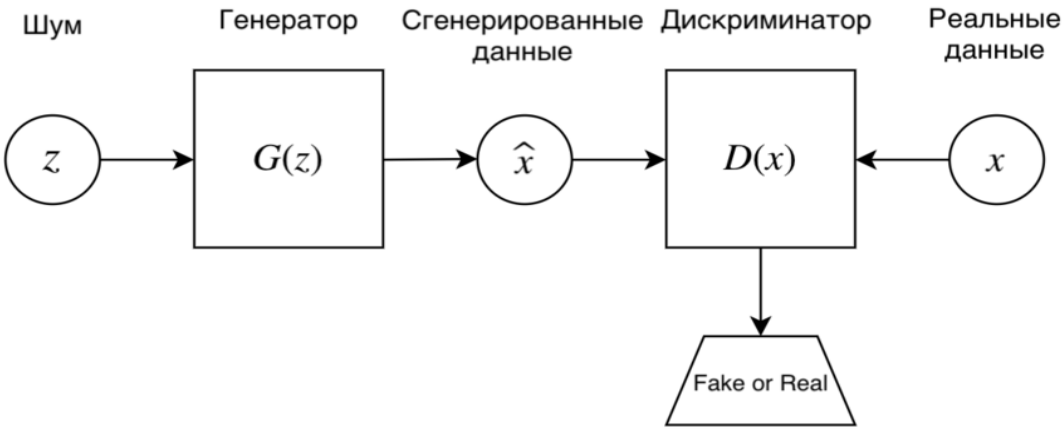

Рис. 1 Оригинальная архитектура GAN

1.3.1.1 Задача

Как было сказано выше архитектура GAN состоит из двух блоков – генератора и дискриминатора, для которых в качестве базовых моделей используются многослойные перцептроны. Тогда, генератор можно представить как отображение $G(z, \gamma_G)$, где G – дифференцируемая функция, представленная многослойным перцептроном с параметром γ_G, а z – шум. Также, чтобы вывести вероятностное распределение генератора p_G над набором входных данных X, определим априорную вероятность шума $p_z(z)$. Подобным образом представим дискриминатор, в виде $D(z, \gamma_D)$, который возвращает вероятность того, что объект x

был получен из исходных входных данных, а не из p_G – сгенерированных данных. Во время обучения дискриминатор стремиться максимизировать вероятность правильной классификации объектов из исходных входных данных и сгенерированных данных. При этом, генератор стремиться минимизировать $\log(1 - D(G(z)))$. Говоря иначе, генератор и дискриминатор «играют» в минимакс игру.

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}}[logD(x)] + E_{z\sim p_z}[\log(1 - D(G(z)))]$$

1.3.1.2 Алгоритм
1.3.1.3 Проблемы обучения

1.3.2 Conditional GAN
1.3.3 DCGAN

Первая статья, где удалось успешно применить свертки в GAN. Важные особенности:

- Не используется Max Pooling, для понижения размерности, вместо этого используем свертки со stride.
- Используем Batch Normalization, как в генераторе (после каждой свертки, кроме последней), так и дискриминаторе (после каждой свертки, кроме первой и последней). Помогает стабилизировать обучение и протекать градиенту в глубоких моделях.
- Не используются полносвязные слои.

Авторы дают ряд рекомендаций, как конструировать сеть:

- Заменить все пулинговые слои на на сверточные со сдвигами (на уровне дискминатора) и на сверточные с дробными сдвигами (на уровне генератора)
- Использовать батч-нормализацию как в генераторе, так и дискриминаторе
- Удалить полносвязнные слои, для увеличения глубины сети
- Использовать ReLU активацию в генераторе для всех слоев, кроме выходного, для которого использовать Tanh.
- Использовать LeakyReLU активации во всех слоях дискриминатора.

А также значения параметров, при которых они обучали модель на разных датасетах:

- Все модели обучались со стохастическим градиентным спуском, где размер батча равен 128.
- Все веса проинициализированы из нормального распределения со стандартным отклонением равным 0.02.
- В LeakyReLU, угол равняется 0.2 во всех моделях
- В качестве оптимизатора использовался Adam со следующими гиперпараметрами: learning rate=0.0002 and momentum term $\beta1=0.5$

Также в статье предлагается использовать обученный дискриминатор как классификатор в задачах обучения с учителем. Подаваемые на вход дискриминатору фичи пропускаются через свертки и на каждом из них применяется max pooling, затем, полученный результат выжимается в вектор, который и выступает как набор признаков для одной картинки. На этом векторе мы уже обучаем модель классификации, авторы использовали SVM.

1.3.4 Improved GAN

Авторы статьи предложили ряд техник, которые, как они заявляют, позволяют сделать процесс обучения GAN лучше. Основная предпосылка заключается в том, что процесс обучения GAN заимствован из теории игр, где игроки (генератор и дискриминатор) пытаются прийти к равновесию Нэша (процесс обучения сети должен быть направлен на сходимость к этому равновесию). Однако используемый градиентный спуск не обязан приводить к достижения этого равновесия. Поэтому авторы предпринимают эвристические попытки приблизиться к нему.

Feature Matching

Идея заключается в том, чтобы заменить функцию ошибки генератора на новую, которая будет предотвращать переобучение генератора на текущем состоянии дискриминатора.

Вместо того, чтобы максимизировать вероятность обмануть дискриминатор, предлагается генерировать такие данные, которые будут близки к статистическим характеристикам реальных данных. Под данными подразумевается feature maps.

И считать в качестве ошибки насколько feature maps полученные от генератора, будут отличаться от ожидаемых значений feature maps полученных по реальным изображениям на одном из промежуточных слоев дискриминатора.

При этом дискриминатор будет использоваться для определения тех самых статистических характеристик, которые действительно важны в данном контексте. Дело в том, что, обучая дискриминатор как обычно, то есть различать реальные объекты от сгенерированных, мы как раз обучаем его слои так, чтобы выделять нужные feature maps, по которым можно хорошо разделять изображения на два упомянутых выше класса.

Говоря иначе, мы как обычно обучаем дискриминатор хорошо разделять два класса, а для этого он как раз выделяет нужные ему feature maps, по которым хорошо видно, чем отличаются два класса. А генератор при этом обучаем так, чтобы он генерировал такие картинки feature maps которых будет максимально близко к feature maps реальных данных, полученных дискриминатором.

Minibatch discrimination

Идея заключается в том, чтобы избежать “коллапса мод”, возникающей по причине того, что дискриминатор смотрит лишь на одно сгенерированное изображение в моменте, то есть он не видит других изображений, которые могут быть порождены генератором при его текущем состоянии. Поэтому предлагается смотреть не только на одно изображение, а сразу не несколько “ближайших” (с точки зрения состояния генератора) к нему.

Дискриминатор смотрит на изображения, полученные от генератора, но смотрит на каждое из них независимо. Поэтому если у генератора произошел коллапс и он генерирует одинаковые (или набор изображений из очень небольшого множества), то дискриминатор не сможет это понять. В том смысле, что как только дискриминатор верно определит, что изображение сгенерировано, генератор на следующей итерации даст ему другое, на котором уже дискриминатор ошибется и это будет происходить в цикле. Именно поэтому мы и хотим получить информацию о других изображениях генератора, чтобы понять, насколько они отличаются друг от друга.

Авторы статьи предлагают сделать это следующим образом:

1. Возьмем feature maps на одном из промежуточных слоев дискриминатора и уложим их в вектор. Так поступим для нескольких картинок, которые нам сгенерировал генератор.
2. Каждый из векторов умножим на некоторый тензор T, в результате получим матрицу M_i.
3. Затем, в каждой из матриц M_i берем некоторую строку с индексом b, например, при b=1 во всех матрицах мы берем первую строку.
4. Зафиксируем значение i и значение b, и для всех оставшихся n объектов, посчитаем L1 норму. То есть, при i=1, b=1 нужно из первой строки матрицы M_1 вычесть поочередно первые строки остальных матриц M_j и подставить эти разницы в L1 норму. После чего взять от них минус и засунуть в экспоненту. После чего, полученные выражения нужно суммировать.
5. Посчитаем полученные величины для всех строк (варьируем b) для одного объекта (фиксируем i). Затем уложим их в вектор.
6. Сделав так для всех объектов (перебрав i), получим матрицу o(X), но она скорее нужна для последующих векторизации вычислений.
7. Полученный вектор $o(x)_i$ мы конкатенируем с входным вектором признаков $f_i(x_i)$

Так как мы подаем дискриминатору не только изображения, полученные от генератора, но и реальные изображения, то описанный выше метод нужно применить отдельно для изображений генератора и реальных изображений, иначе это просто потеряет всякий смысл. То есть, оценивать похожесть между собой сгенерированных и реальных изображений, взятых вперемешку - не очень информативно. Потому что, реальные изображения имеют свою некоторую степень похожести, к которой, по-хорошему, сгенерированные изображения должны стремиться.

Также полезно напомнить, что мы подаем дискриминатору на вход один объект, но теперь этот объект хранит в себе информацию о “близких” с ним изображениях.

Данный подход по заявлению авторов, оказывается мощнее чем ***feature matching*** и позволяет генерировать более привлекательные визуально образцы. При

этом, если мы хотим натренировать классификатор, то наоборот, ***feature matching*** оказывается сильнее.

Historical Averaging

Идея заключается в том, чтобы усреднять параметры сети по их историческим значениям, говоря иначе, считать их скользящей средней. Авторы показывают на модельных примерах из теории игр, что такой подход помогает выйти из ситуацию, так как сам по себе градиентный спуск не позволяет прийти к точке равновесия.

One-sided label smoothing

Предлагается заменить метки таргета, которыми оперирует дискриминатор с бинарных, на более сглаженные. Положительные заменяются на α отрицательные на β. При этом, авторы предлагают использовать сглаживание только для положительных классов, то использовать только α, а для отрицательных не использовать β, а как обычно использовать 0 как метку класса.

Virtual batch normalization

Сам по себе Batch Normalization прекрасен и очень хорошо зарекомендовал себя при обучении GAN - в том же DCGAN. Однако у него есть недостаток, заключающийся в том, что итоговый выход сети может сильно зависеть от того, какие входные данные попали в батча. Поэтому предлагается, избавиться от такой подвижности и зафиксировать некоторый эталонный батча, по которому посчитать среднее и стандартное отклонение. Важно, что этот эталонный батч выбирается одни раз перед началом обучения и фиксируется. То есть, мы будем из нормировать данные внутри батчей не по характеристикам их батча, а по характеристикам эталонного батча.

Также авторы утверждают, что данная процедура вычислительно дорогая, так как нужно гнать сразу два батча через сеть, поэтому они используют его только в генераторе.

1.3.5 SGAN
1.3.6 BigGAN
1.3.7 StyleGAN2-ADA
1.3.8 VAE