Санкт–Петербургский государственный университет

МИРОШНИЧЕНКО Александр Сергеевич

Отчет по научно-исследовательской работе

Улучшение качества изображений с помехами с использованием генеративных состязательных сетей

Уровень образования: бакалавриат
Направление 01.03.02 «Прикладная математика и информатика»
Основная образовательная программа СВ.5005.2017 «Прикладная математика, фундаментальная информатика и программирование»
Профиль «Исследование и проектирование систем управления и обработки сигналов»

Научный руководитель:
доцент, кафедра теории систем управления электрофизической аппаратурой,
к.ф. - м.н. Козынченко Владимир Александрович

Санкт-Петербург
2020 г.

Содержание

Введение

При съемке фотографий могут возникнуть различные искажения. Например, фотография может получится размытой. Произойти это может, если матрица фотоаппарата неисправна, или если объектив камеры движется слишком быстро и не успевает сфокусироваться. В таких случаях можно повторить снимок, но какие-то объекты, которые были засняты в первый раз, уже могут и не появиться в кадре. Иногда фотографию и вовсе нельзя переснять.

Встает вопрос о восстановлении качества таких изображений с помехами. При этом надо понимать, что, конечно, мы не сможем восстановить именно ту фотографию, которая могла получиться, не будь обстоятельств, смазавших кадр. Ведь, по сути дела, можно представить искажение четкой фотографии как некоторое отображение из множества четких фото в множество размытых. И такое отображение с большой долей вероятности не будет биективным. То есть наверняка существует много различных четких фотографий, которые при обратном отображении в множество размытых, дают один и тот же результат. Однако можно попытаться сказать, каким мог бы быть один из возможных вариантов таких четких изображений. Для такой проблемы можно использовать нейронные сети, так как их решают задачу перевода одного вектора данных в другой.

Постановка задачи

Написать нейронной сеть, улучшающую качество размытых изображений при помощи языка Python 3.8 и программной библиотеки для машинного обучения TensorFlow 2.3.0.

Метод обучения

Среди существующих видов сетей можно отметить GAN - генеративные состязательные сети. Модели GAN обучаются отображению случайного вектора шума z в изображение y, G : $z \to y$. Условные же GAN учатся преобразованию изображения x и случайного вектора шума z в изображение y, G : $\{x, z\} \to y$. Генератор G обучается генерировать изображения, которые не может отличить от «подлинных» дискриминатор D, который обучается распознавать «подделки» генератора. Этот процесс изображен на рисунке 1. Целевая функция условной GAN может быть выражена как:

$$loss_{\text{GAN}} = E_{x,y} \left[\log D\left(x, y\right)\right] + E_{x,z} \left[\log\left(1 - D\left(x, G\left(x, z\right)\right)\right)\right],$$

где G пытается минимизировать *loss*, в то время как соревнующийся с ним D пытается максимизировать ее. Другими словами $\text{G}^* = \arg\min_G \max_D loss_{\text{GAN}}\left(G, D\right)$
В статье о сети pix2pix было проведено исследование, в котором функция L1 в сочетании с традиционной $loss_{\text{GAN}}$ приводила к тому, что работа дискриминатора D не изменялась, а вот генератор G обучался не только обманывать дискриминатор, но и генерировать изображения, близкие к истинным.

$$L1\left(G\right) = E_{x,y,z}\left[\|y - G\left(x, z\right)\|\right]$$

В итоге, наша функция выглядит как:

$$G^* = \arg\min_G \max_D loss_{\text{GAN}}\left(G, D\right) + \lambda L1\left(G\right)$$

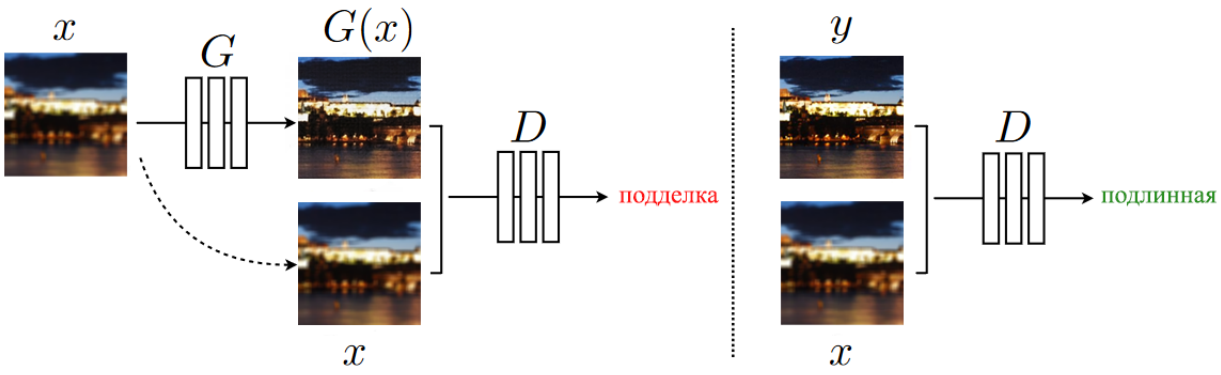

Рис. 1: Генератор G обучается генерировать изображения, которые не может отличить от «подлинных» дискриминатор D, который обучается распознавать «подделки» генератора.

Значение $\lambda = 100$, так как именно это значение предлагают авторы вышеупомянутой статьи. Алгоритм обучения представлен на рисунке 2. Датасет изначально перемешан и разбит на батчи. В данном случае размер батча равен одному. Прогон каждого батча и изменение весов сети обозначены одним шагом обучения.

Подготовка данных для обучения

Для того, чтобы обучить нейронную сеть, необходимо иметь как размытые изображения *x*, так и четкие изображения *y*. Четкие изображения *y* было решено взять из открытого датасета places365, состоящего из 365 категорий четких фотографий, собранных из интернета. Из этих категорий были выбраны следующие 4: замки, леса, горы и небоскребы. Для тренировочного датасета из каждой категории было выбрано по 250 фотографий, а для тестового - 150. Каждая фотография была обрезана до размера 256×256 пикселей. На каждую фотографию был наложен эффект размытия при помощи функции cv2.blur() с ядром $(14, 14)$ из библиотеки OpenCV. Это имитация реальных помех, которые могут произойти при съемке фотографий. Всего получилось 2000 изображений (1000 четких и 1000 размытых) для тренировочного датасета и 1200 изображений (600 четких и 600 размытых) для тестового датасета. Они были сохранены в 4 файлах: *train_precise_1000.npy, train_blurred_1000.npy, test_precise_600.npy, test_blurred_600.npy*
Затем файлы были загружены на облако Google Drive.
Код скрипта, который обрабатывает выбранные самим пользователем фотографий из уже скачанного и разархивированного датасета places365 находится в файле prepare_data.ipynb.

Реализация

Задачу было принято решать в среде Google Collaboration при помощи IPython 3.8. В интерактивном режиме намного проще отслеживать прогресс обучения сети и анализировать результаты обучения. Также в Google Collaboration процесс обучения проходит гораздо быстрее чем нам CPU персонального компьютера. Google предоставляет GPU Nvidia Tesla K40. А GPU, как

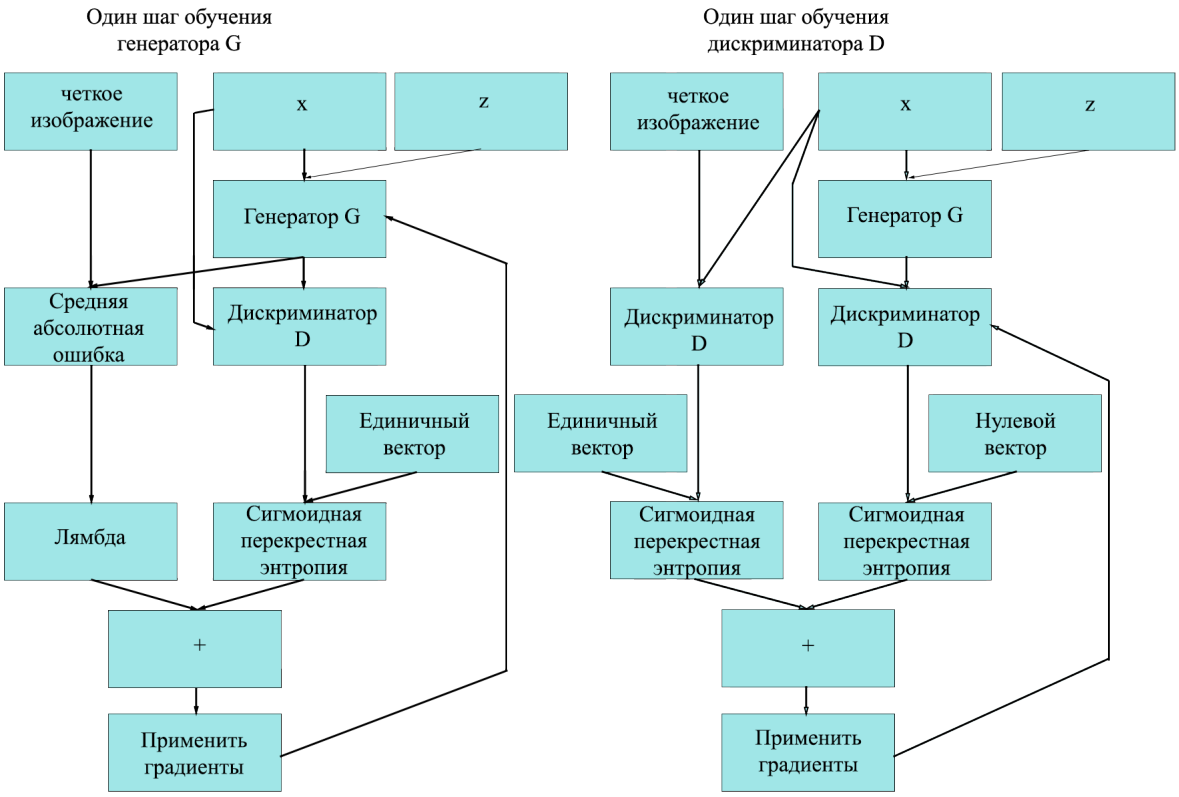

Рис. 2: Шаг обучения для генератора и дискриминатора

известно, специализируется на операциях с векторами и быстрее в плане вычислений такого типа.

Код основной программы представлен в файле deblurrr.ipynb.

Структура программы следующая:

1. Примонтирование Google Drive директории с датасетами.
2. Загрузка датасетов.
3. Просмотр пяти изображений из загруженных датасетов для проверки корректной загрузки данных.
4. Класс, описывающий генератор.
5. Целевая функция генератора.
6. Класс, описывающий дискриминатор.
7. Целевая функция дискриминатора.

8. Функция одного шага обучения.
9. Основной цикл обучения.
10. Просмотр результатов.

Структуры генератора и дискриминатора приведены в коде программы. Количество обучаемых параметров сети - 57 183 620.

Изначально количества эпох было равно 150 с целью проверить скорость сходимости. Обучение заняло 6 часов. Было отмечено, что сеть обучилась решать задачу примерно за 40 эпох. Поэтому количество эпох было принято сократить до 40. Время, потраченное на их обучение, составило 1 час.

Результаты обучения представлены в коде программы. Пример работы обученной нейронной сети приведен на рисунке 3.

Рис. 3: Пример работы обученной сети

Заключение

Получена реализация алгоритма улучшения качества размытых изображений. Выбранный способ реализации позволяет легко просмотреть поэтапно, из чего состоит алгоритм, а так же в режиме реального времени следить за процессом обучения сети, и в случае, к примеру, с более высоким уровнем размытия входных изображений быстро определить, на какой эпохе стоит остановить обучение.

Список литературы

[1] Aurélien G. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems / Aurélien G. — 2nd Edition — O'Reilly Media, 2019.

[2] Kailash A. Generative Adversarial Networks Projects: Build next-generation generative models using TensorFlow and Keras / Kailash A. — Packt Publishing, 2019.

[3] Документация TensorFlow [Электронный ресурс]. — Режим доступа: https://www.tensorflow.org/api_docs/python/tf

[4] Портал ML Glossary [Электронный ресурс]. — Режим доступа: https://ml-cheatsheet.readthedocs.io

[5] Курс на платформе Coursera [Электронный ресурс]. — Режим доступа: https://www.coursera.org/learn/getting-started-with-tensor-flow2

Приложение: код программы

Ссылка на репозиторий с кодом: `https://github.com/polotent/deblurrr`