

# 1 Описание реализованных архитектур

В данном домашнем задании я реализовал модель pix2pix. В ходе экспериментов данная модель обучалась с функцией потерь L1, а затем и с функциями потерь conditionalGAN (предложенной авторами статьи, далее cGAN) и L1+cGAN.

Я использовал в модель, описанную в статье [pix2pix](#) (далее - статья). Она состоит из двух основных частей: генератора и дискриминатора. В качестве генератора была использована архитектура U-Net, а в качестве дискриминатора архитектура PatchGan. Рассмотрим их подробнее:

## 1 U-Net

Архитектура U-Net представляет из себя энкодер и декодер со скайл-коннекшнами между каждым блоком энкодера и каждым блоком декодера. Я использовал архитектуру состоящую из 7 блоков в энкодере и 7 блоков в декодере. Каждый из блоков представляет из себя 4 слоя:

- a) Свертка (или траспоуз-свертка) с ядром 4x4, страйдом 2x2, паддингом 1x1
- b) Батч-нормализация (или ее отсутствие)
- c) Слой дропаута (или его отсутствие) с вероятностью 0.5
- d) Функция активации

Как и описано в статье, для блоков энкодера используются: свертка, батч-нормализация и функция активации LeakyReLU с параметром 0.2 (это значение будем считать стандартным во всей работе в дальнейшем). Для блоков декодера используются: траспоуз свертка, батч-нормализация, слой дропаута (для первых четырех блоков) и функция активации ReLU. Последний блок декодера не использует батч-нормализацию и вместо ReLU применяется Tanh.

Количество фильтров также совпадает со статьей и представляет из себя следующую последовательность из out\_channels:

64-128-256-512-512-512-512 для энкодера

512-512-512-256-128-64-3 для декодера

Для декодера количество входных каналов равно количеству `out_channels*2` везде, кроме первого блока, где они равны (из-за скип-коннекшнов между блоками энкодера и декодера).

Итого генератор имеет 54.4 миллиона параметров.

## 2 PatchGAN Classifier 70x70

Данная архитектура состоит из 5 блоков вида:

- a) Свертка с ядром 4x4, страйдом 2x2 (или 1x1), паддингом 1x1
- b) Батч-нормализация (или ее отсутствие)
- c) Функция активации

Для первого блока используется только свертка и LeakyReLU, для последующих трех слоев используются свертки, удваивающие количество каналов, батч-нормализации и LeakyReLU. Четвертый и пятый слои используют страйд 1x1. Последний(пятый) слой также использует свертку с `out_channels=1` и сигмоидную функцию активации (в реализации вместо сигмоиды используется BCEWithLogitsLoss для поддержания численной стабильности).

Количество фильтров также совпадает со статьей и представляет из себя следующую последовательность из `out_channels`:

64-128-256-512-1

Дискриминатор применяется к шестиканальному тензору, сконкатенированному из двух трехканальных тензоров (картиночка, в случае `batch_size=1`). На выходе имеется тензор размера (`batch_size, 30, 30, 1`).

Итого дискриминатор имеет 2.7 миллиона параметров.

Для валидации и инференса генератор работает в режиме обучения, то есть батч-нормализации высчитываются для каждого нового поступившего батча, а также применяется `dropout`. Это делается для введения шума в модель, как и описывают авторы статьи.

## 2 Гиперпараметры

### 2.1 Наборы данных и гиперпараметры моделей

Я провел обучение и тестирование вышеописанной модели на двух датасетах: facades и day2night. В первом случае модель использовалась для перевода сегментации в настоящий фасад, а во втором – перевода дневного фото в ночное. Сначала опишу общие, базовые гиперпараметры:

- set\_seed(21) для фиксирования всего, что только возможно
- $\lambda = 100$
- batch\_size=1 для facades и batch\_size=4 для day2night
- Обучение проводилось 200 эпох (и 17 эпох для day2night) с оптимайзером Adam, betas=(0.5, 0.999) и LR=0.0002. По времени это порядка нескольких часов (зависит от GPU).
- Для трекинга процесса использовался [мой wandb](#). Там есть графики и примеры работы, но в коде его использование за-комментировано.

### 2.2 Обучение

Обучение в базовом варианте проводится как описано в статье. На каждом шаге обучения выполняются следующие этапы:

- 1) Генерация изображения генератором:  $G_x = G(x)$
- 2) Оптимизация параметров дискриминатора:  
$$\text{loss} = (\text{CE}(D(x, G_x), \text{false}) + \text{CE}(D(x, y), \text{true})) / 2$$
- 3) Оптимизация параметров генератора:  
$$\text{loss} = \text{CE}(D(x, G_x), \text{true}) + \lambda \text{L1}(D(x, G_x), y)$$

### 2.3 Тонкости реализации

После некоторых(более десятка) неудачных попыток обучить модель так, как описано в статье, я обратился к предложенному в задании репозиторию для выяснения деталей реализации. Я запустил код репозитория и узнал следующие детали:

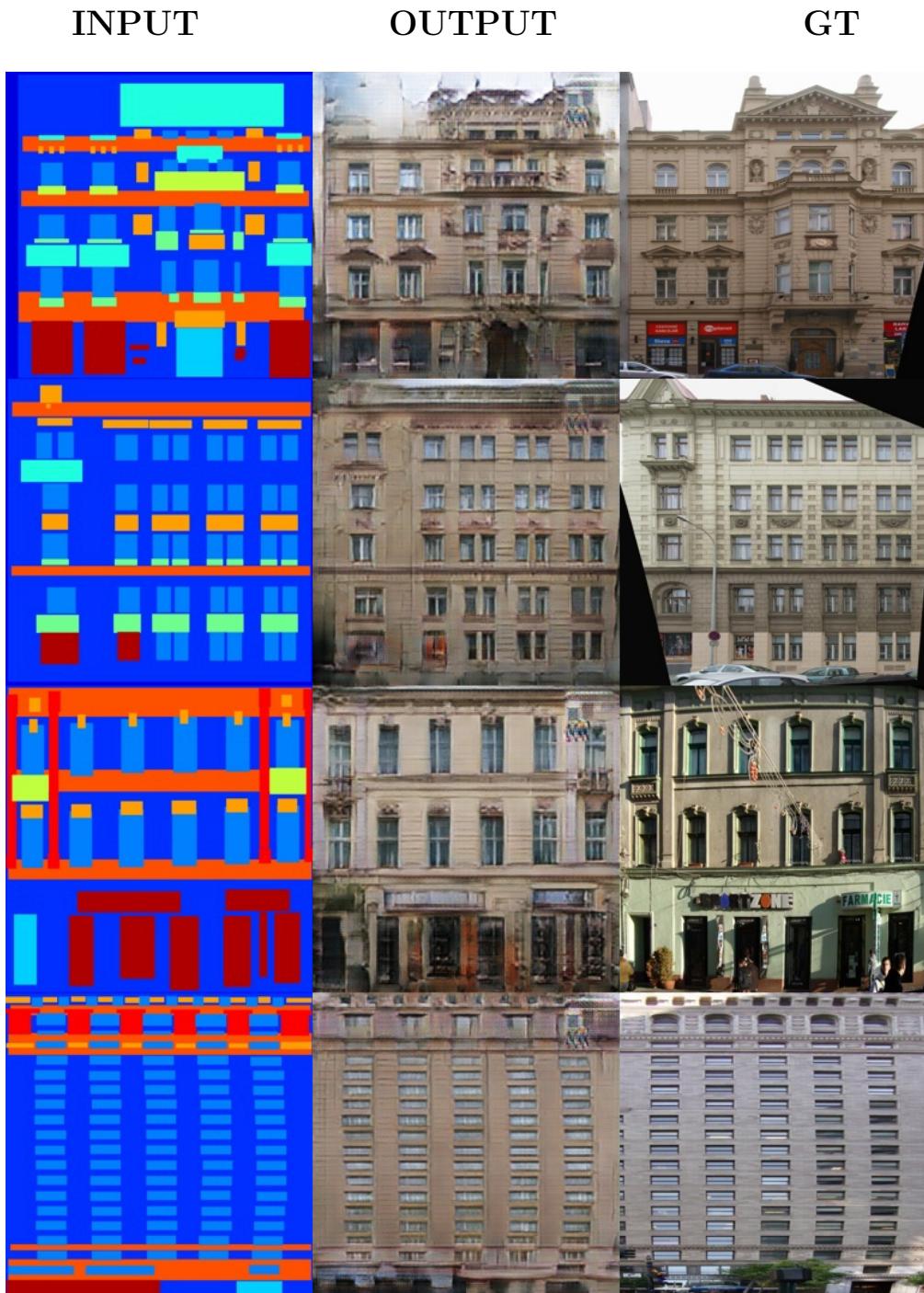
- Инициализация весов сети. Веса сверток и линейных слоев инициализировались нормальным распределением с параметрами 0; 0.02. Все bias инициализировались нулями.
- Параметры батч-нормализаций.  $\text{eps}=1e-05$ ,  $\text{momentum}=0.1$ ,  $\text{affine}=\text{True}$ .
- Наличие/отсутствие bias. Везде отсутствует, кроме первого блока в генераторе, и первого и последнего блока в дискриминаторе.
- Аугментации. Аугментации вида: `resize` при помощи бикубической интерполяции до 286x286, `crop` случайного места до 256x256, отображение относительно вертикальной оси с вероятностью 0.5.

Все эти детали реализованы мной в коде. Также еще стоит отметить нормализацию картинок для перевода значений каждого пикселя от  $[0, 1]$  в  $[-1, 1]$ .

### 3 Эксперименты

#### 3.1 Примеры финальных результатов

Стоит для начала продемонстрировать качественные результаты на обоих наборах данных (как я сделал бы, будь это статья):



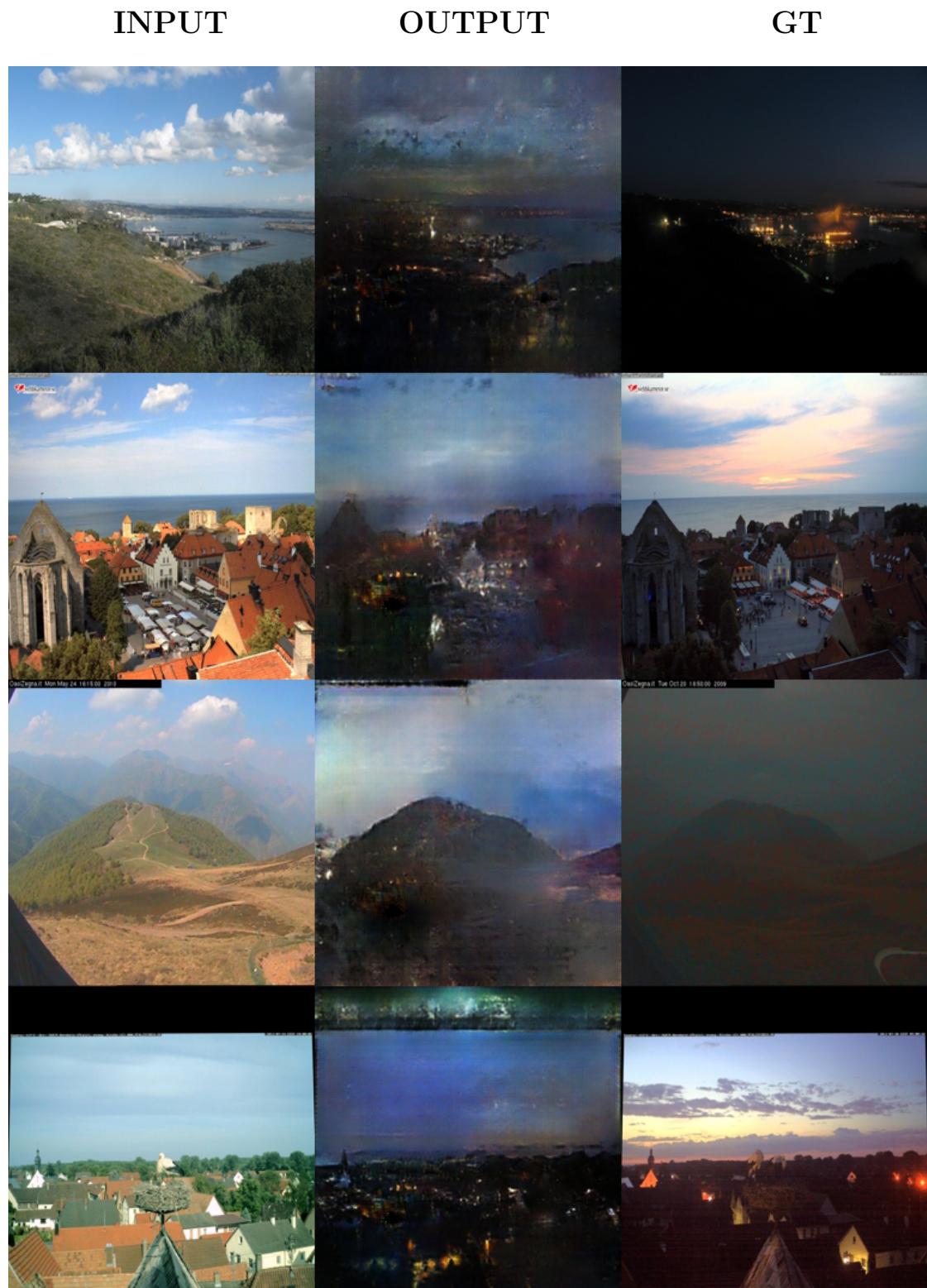


Рис. 3.2: Хорошие примеры работы модели на датасете day2night

статье pix2pix на этом наборе данных, то там результаты не сильно лучше.

## 3.2 Основные эксперименты

Далее расскажу об экспериментах и том, как я получил такие результаты. В основном эксперименты проводились на наборе данных facades, так как он меньше. Если не указано иного, то эксперимент проводился на датасете facades.

Главный эксперимент был в добавлении аугментаций. Гипотеза заключалась в том, что это поможет от переобучения и даст лучшие результаты. Были применены аугментации описанные в статье, а именно: resize, crop, случайное отображение относительно вертикальной оси. Посмотрим на как на некоторую числовую метрику на значение лосса генератора (которое учитывает макс. правдоподобие):

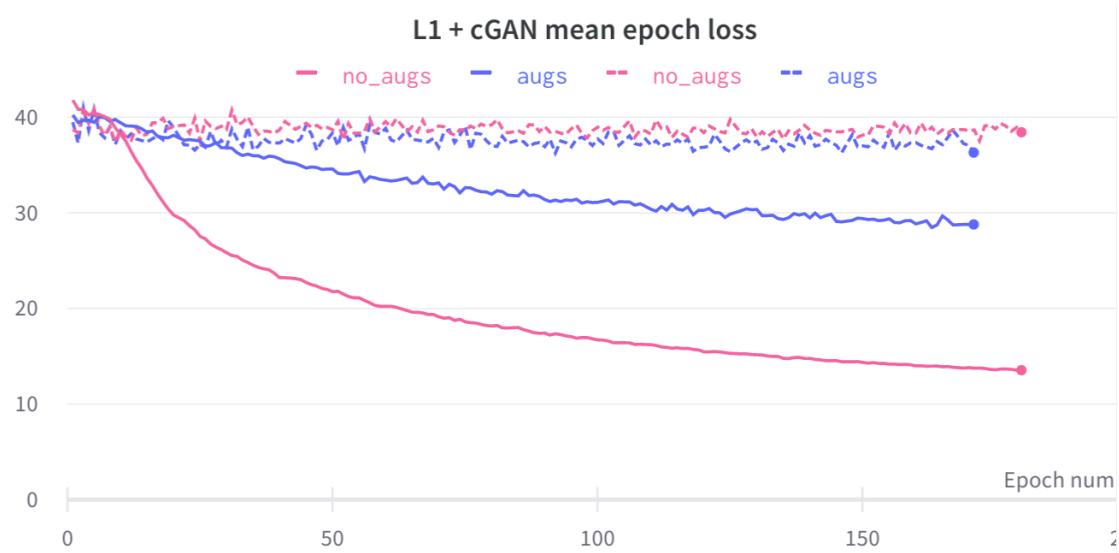


Рис. 3.3: Сравнение L1+cGAN loss для моделей с и без аугментации (прерывистая линия - валидационная выборка)

Хоть это и не идеальная метрика, но в данном случае хорошо видно, что добавление аугментаций справляется с переобучением. Лосс на валидационной выборке уменьшается при добавлении аугментаций, хотя на обучающей выборке сильно выше. Также сравним результаты качественно:

INPUT

OUTPUT

GT

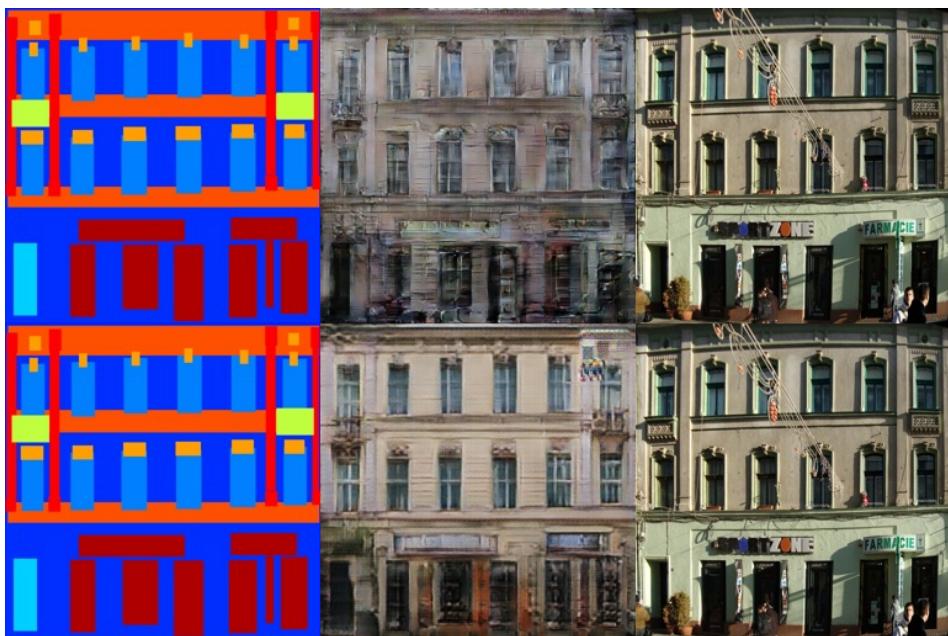


Рис. 3.4: Качественное сравнение для моделей без и с аугментациями на валидации. Первый ряд - без аугментаций, второй - с аугментациями.

Можно заметить, как фасад стал менее размытый, появилось меньше артефактов по углам картинки.

Также я провел эксперименты с различными значениями  $\lambda$  это 100, 125 и 10:



Рис. 3.5: Качественное сравнение с различными занчениями  $\lambda$  (100, 125, 10) в процессе обучения

На рис. 3.5 видно, что  $\lambda=10$  сохраняет общую форму лучше,  $\lambda = 125$  делает картинку более четкой, но имеет артефакты (в районе окон, первый этаж), а  $\lambda=100$  имеет более сбалансированный результат, что подтверждает нашу гипотезу. Сравнение потерь генератора в данном случае непоказательно, т.к. зависит от  $\lambda$ , поэтому в данном случае пришлось опираться только на качественное

сравнение.

Также были и неудачные эксперименты. Гипотеза: уменьшение  $\lambda$  ведет к большему доминированию cGAN функции потерь, то есть должно давать лучше структуру и детали, но их месторасположения могут быть нарушены (за это отвечает L1 функция потерь). При попытке менять  $\lambda$  для датасета day2night получались такие артефакты:



Рис. 3.6: Результаты работы модели на датасете day2night с  $\lambda=50$

Если присмотреться, то можно заметить, что в центре каждой картинки сохраняются аналогичные артефакты. Я нашел об этом информацию в интернете: возможно, это связано с параметром  $\lambda$  или аугментациями. На некоторых примерах картинка и правда стала четче, но из-за артефакта в центре непригодна.

Также были проведены и другие незначительные эксперименты (напр. с размерами генератора и дискриминатора), однако, они не увенчались успехом.

### 3.3 Поиск метрики для сравнения с бейзлайном

Для реализации численной метрики для количественного сравнения (помимо функции потерь генератора) был выбран индекс структурного сходства ([SSIM](#), structure similarity). Т.к. это один из методов измерения схожести между двумя изображениями. Также он учитывает взаимосвязь рядомстоящих пикселей (через ковариацию) в отличие от MSE и PSNR. В реализации подсчет метрики усреднялся по каналам. С данной метрикой был проведен базовый эксперимент: сравнение модели без adversarial функции потерь (только L1) и с ней (L1+cGAN). Проведем количественное сравнение данных моделей:

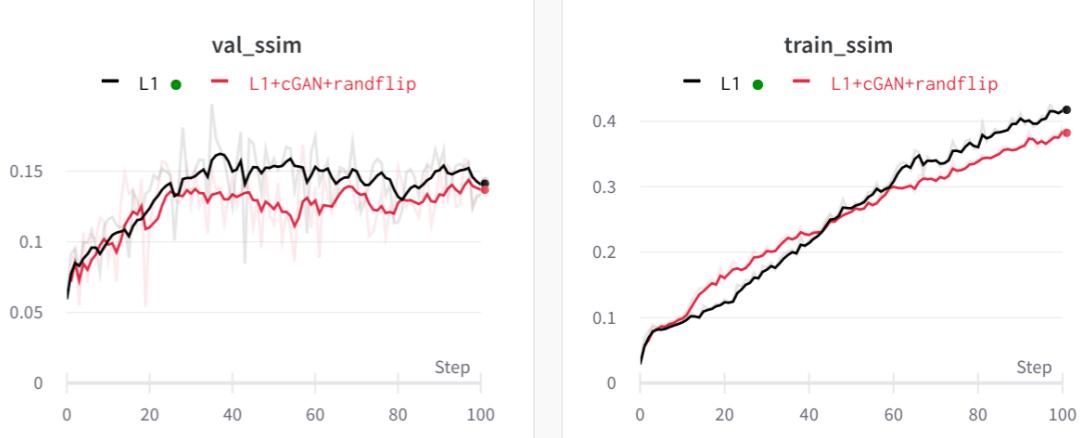
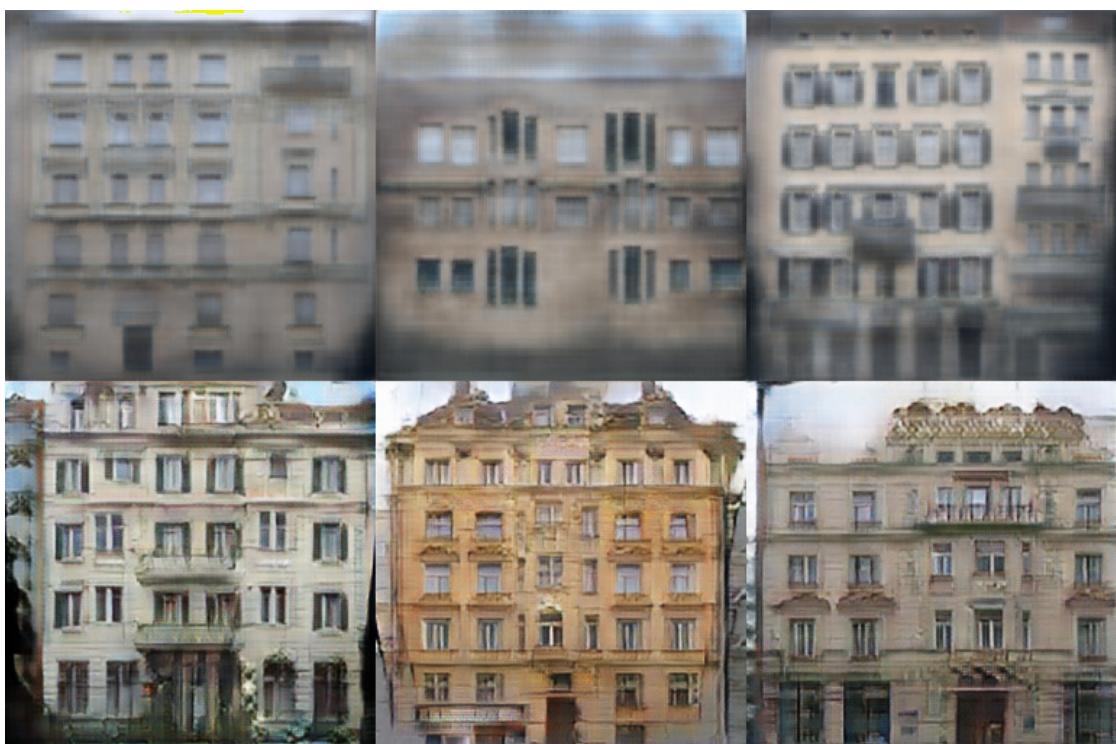


Рис. 3.7

Можно заметить, что модель с функцией потерь L1 имеет большую структурную схожесть сгенерированных картинок с GT картинками. Однако давайте теперь проведем качественное сравнение данных моделей:

Рис. 3.8: 1 строка: L1, SSIM=0.42



2 строка: L1+cGAN, SSIM=37

Как несложно заметить, результаты схожи с результатами авторов статьи:



Рис. 3.9: Качественное сравнение авторами статьи L1 и L1+cGAN функций потерь (см. 2 строку)

Из данного сравнения и графика можем сделать вывод, что на самом деле метрика SSIM не совсем подходит для измерения качества на нашей задаче. Если рассмотреть данную метрику более подробно, то можно найти этому объяснение: L1 стремится усреднить значения и сделать картинку в целом более похожей, в то время как cGAN функция потерь считается на основе дискриминатора и стремится сделать состоящей из настоящих патчей.

Для сравнения моделей с небольшими изменениями я использовал значения L1+cGAN функции потерь на валидационной выборке. При серьезных изменениях(архитектуры, batch\_size и др.) думаю, что стоит руководствоваться качественным сравнением некоторых результатов на валидационной выборке.