

ВАРИАЦИОННЫЕ АВТОКОДИРОВЩИКИ

Сергей Николенко

Академия MADE – Mail.Ru

01 декабря 2021 г.

Random facts:

- 1 декабря каждого года каракалпаки всего мира отмечают День каракалпакского языка
- 1 декабря 1822 г. был провозглашён императором основатель Бразильской империи дон Педру I (Педру ди Алкантара Франсишку Антониу Жуан Карлуш Шавьер де Паула Мигел Рафаэл Жоаким Жозе Гонзага Пашкуал Киприану Серафим де Браганса и Бурбон)
- 1 декабря 1887 г. вышла повесть «Этюд в багровых тонах» (первая книга о Шерлоке Холмсе), а 1 декабря 1903 г. на экраны вышло «Большое ограбление поезда»
- 1 декабря 1962 г. Никита Хрущёв посетил выставку художников-авангардистов студии «Новая реальность» в Манеже; будучи неподготовленным к восприятию абстрактного искусства, генсек подверг его, гм, резкой критике; впрочем, уже через два года Хрущёв признал, что «не совсем разобрался», а 1 декабря 2006 г. в ЮАР впервые на африканском континенте вступил в силу закон об однополых браках
- 1 декабря 1959 г. в Вашингтоне был заключён Договор об Антарктике, который предусматривает свободу научных исследований и запрещает любые ядерные взрывы и захоронения радиоактивных материалов

ВАРИАЦИОННЫЕ АВТОКОДИРОВЩИКИ

БАЙЕСОВСКИЙ ВЫВОД В НЕЙРОННЫХ СЕТЯХ

- Для определённости будем говорить о задаче классификации, $D = (X, Y)$ из пар вида (\mathbf{x}, y) .
- Идея такая же, как всегда: найти апостериорное распределение

$$p(\mathbf{w} \mid X, Y) = \frac{p(Y \mid X, \mathbf{w})p(\mathbf{w})}{p(Y \mid X)} \propto p(Y \mid X, \mathbf{w})p(\mathbf{w})$$

и предсказательное распределение

$$\begin{aligned} p(y \mid \mathbf{x}, X, Y) &= \int_{\mathbf{w}} p(y \mid \mathbf{x}, \mathbf{w})p(\mathbf{w} \mid X, Y)d\mathbf{w} \propto \\ &\propto \int_{\mathbf{w}} p(y \mid \mathbf{x}, \mathbf{w})p(Y \mid X, \mathbf{w})p(\mathbf{w})d\mathbf{w}. \end{aligned}$$

- Но $p(\mathbf{w} \mid X, Y)$ имеет очень сложный вид, и аналитически ничего не найдёшь.
- Нужны приближения.

БАЙЕСОВСКИЙ ВЫВОД В НЕЙРОННЫХ СЕТЯХ

- *Вариационные приближения*: приближаем сложное распределение более простым, выбирая то из более простого семейства, которое лучше всего приближает сложное распределение.
- «Похожесть» можно определить по расстоянию Кульбака–Лейблера:

$$\text{KL}(p\|q) = \int p(x) \ln \frac{p(x)}{q(x)} dx = - \int p(x) \ln \frac{q(x)}{p(x)} dx.$$

БАЙЕСОВСКИЙ ВЫВОД В НЕЙРОННЫХ СЕТЯХ

- Чтобы максимизировать $p(X)$ со скрытыми переменными Z , где $p(Z | X)$ – сложное распределение, ищем приближение $q(Z)$:

$$\ln p(X) = \mathcal{L}(q) + \text{KL}(q \| p(Z | X)), \text{ где}$$

$$\mathcal{L}(q) = \int_Z q(Z) \ln \frac{p(X, Z)}{q(Z)} dZ, \quad \text{KL}(q \| p) = - \int_Z q(Z) \ln \frac{p(Z | X)}{q(Z)} dZ.$$

- И мы можем максимизировать нижнюю оценку $\mathcal{L}(q)$, приближая $q(Z)$ к $p(Z | X)$.

БАЙЕСОВСКИЙ ВЫВОД В НЕЙРОННЫХ СЕТЯХ

- Для нейронных сетей тоже вводим $q(\mathbf{w})$ и пытаемся приблизить им $p(\mathbf{w} \mid X, Y)$, минимизируя расстояние Кульбака–Лейблера между ними:

$$\begin{aligned}\text{KL}(q(\mathbf{w})\|p(\mathbf{w} \mid X, Y)) &= \int q(\mathbf{w}) \log \frac{q(\mathbf{w})}{p(\mathbf{w} \mid X, Y)} d\mathbf{w} = \\ &= \int q(\mathbf{w}) \log \frac{q(\mathbf{w})}{p(\mathbf{w})p(Y \mid X, \mathbf{w})} d\mathbf{w} + \text{const} = \\ &= - \int q(\mathbf{w}) \log p(Y \mid X, \mathbf{w}) d\mathbf{w} + \int q(\mathbf{w}) \log \frac{q(\mathbf{w})}{p(\mathbf{w})} d\mathbf{w} + \text{const} = \\ &= - \int q(\mathbf{w}) \log p(Y \mid X, \mathbf{w}) d\mathbf{w} + \text{KL}(q(\mathbf{w})\|p(\mathbf{w})) + \text{const} = \\ &= - \sum_{i=1}^N \int q(\mathbf{w}) \log p(y_i \mid f_{\mathbf{w}}(\mathbf{x}_i)) d\mathbf{w} + \text{KL}(q(\mathbf{w})\|p(\mathbf{w})) + \text{const},\end{aligned}$$

БАЙЕСОВСКИЙ ВЫВОД В НЕЙРОННЫХ СЕТЯХ

- Первая ласточка – (Hinton, van Camp, 1993) с очень сильным предположением на $q(\mathbf{w})$:

$$q(\mathbf{w}) = \prod_{w \in \mathbf{w}} q_{\mu_w, \sigma_w}(w) = \prod_{w \in \mathbf{w}} N(w \mid \mu_w, \sigma_w).$$

- Но и это оказалось нелегко:
 - аналитически только с одним скрытым уровнем;
 - без корреляций между весами получается плохо;
 - а с ними алгоритмы вывода становятся квадратичными от числа весов, что не работает.

БАЙЕСОВСКИЙ ВЫВОД В НЕЙРОННЫХ СЕТЯХ

- Две основные сложности:
 - $\int q(\mathbf{w})p(y_i | f_{\mathbf{w}}(\mathbf{x}_i))d\mathbf{w}$ никак не вычислить, если у сети больше одного скрытого уровня;
 - даже чтобы подсчитать функцию, которую мы минимизируем, нужно взять сумму по всему датасету.
- Вторую проблему решить несложно: выберем случайное подмножество S размера $M < N$ и перевзвесим:

$$\mathcal{L}(\mathbf{w}) = -\frac{N}{M} \sum_{i \in S} \int q(\mathbf{w})p(y_i | f_{\mathbf{w}}(\mathbf{x}_i))d\mathbf{w} + \text{KL}(q(\mathbf{w}) \| p(\mathbf{w})).$$

- Если S выбирать случайно, получится несмещённая оценка исходной суммы (как стохастический градиентный спуск).

БАЙЕСОВСКИЙ ВЫВОД В НЕЙРОННЫХ СЕТЯХ

- Чтобы оценить интеграл, заметим, что надо оценить

$$I(\theta) = \frac{\partial}{\partial \theta} \int f(x)p_\theta(x)dx.$$

- Трюк с репараметризацией: пусть $p_\theta(x)$ можно параметризовать как $p(\epsilon)$ уже без параметров, где $x = g(\theta, \epsilon)$.
- Например, для нормального распределения: если $p_\theta(x) = N(x | \mu, \sigma^2)$, то $g(\theta, \epsilon) = \mu + \sigma\epsilon$, и $p(\epsilon) = N(\epsilon | 0, \mathbf{I})$.

БАЙЕСОВСКИЙ ВЫВОД В НЕЙРОННЫХ СЕТЯХ

- Тогда можно получить (сейчас не будем) несмешённую оценку

$$\frac{\partial}{\partial \theta} \int f(x)p_{\theta}(x)dx \approx \frac{\partial}{\partial \theta} f(g(\theta, \epsilon)) = f'(g(\theta, \epsilon)) \frac{\partial}{\partial \theta} g(\theta, \epsilon).$$

- Например, для нормального распределения, подставляя $x = \mu + \sigma\epsilon$, мы получим

$$\frac{\partial}{\partial \mu} \int f(x)p_{\theta}(x)dx = \int f'(x)p_{\theta}(x)dx,$$

$$\frac{\partial}{\partial \sigma} \int f(x)p_{\theta}(x)dx = \int f'(x) \frac{x - \mu}{\sigma} p_{\theta}(x)dx.$$

БАЙЕСОВСКИЙ ВЫВОД В НЕЙРОННЫХ СЕТЯХ

- И теперь подставим обратно в целевую функцию $\mathcal{L}(\mathbf{w})$:
сделаем репараметризацию

$$\begin{aligned}\mathcal{L}(\theta) = & -\frac{N}{M} \sum_{i \in S} \int q_\theta(\mathbf{w}) \log p(y_i \mid f_{\mathbf{w}}(\mathbf{x}_i)) d\mathbf{w} + \text{KL}(q(\mathbf{w}) \| p(\mathbf{w})) = \\ & -\frac{N}{M} \sum_{i \in S} \int p(\epsilon) \log p(y_i \mid f_{g(\theta, \epsilon)}(\mathbf{x}_i)) d\epsilon + \text{KL}(q(\mathbf{w}) \| p(\mathbf{w}))\end{aligned}$$

и подставим стохастическую оценку вместо интеграла:

$$\hat{\mathcal{L}}(\theta) = -\frac{N}{M} \sum_{i \in S} \log p(y_i \mid f_{g(\theta, \epsilon)}(\mathbf{x}_i)) + \text{KL}(q(\mathbf{w}) \| p(\mathbf{w})).$$

БАЙЕСОВСКИЙ ВЫВОД В НЕЙРОННЫХ СЕТЯХ

- Один шаг минимизации расхождения между $q_\theta(\mathbf{w})$ и $p(\mathbf{w} \mid X, Y)$:
 - сначала сэмплировать M случайных примеров из тренировочного набора размера N и M случайных величин $\hat{\epsilon}_i \sim p(\epsilon)$;
 - затем вычислить обновление весов как

$$\Delta\theta = -\frac{N}{M} \sum_{i \in S} \frac{\partial}{\partial\theta} \log p(y_i \mid f_{g(\theta, \hat{\epsilon}_i)}(\mathbf{x}_i)) + \frac{\partial}{\partial\theta} \text{KL}(q(\mathbf{w}) \| p(\mathbf{w})).$$

БАЙЕСОВСКИЙ ВЫВОД В НЕЙРОННЫХ СЕТЯХ

- Постепенно обновляя θ в распределении $q_\theta(\mathbf{w})$, мы приближаемся к истинному апостериорному распределению $p(\mathbf{w} | X, Y)$ – то, что надо!
- Потом можно будет и байесовские предсказания делать: сэмплировать несколько разных наборов весов $\hat{\mathbf{w}}_r \sim q_\theta(\mathbf{w})$ и усреднить результаты как

$$q_\theta(y | \mathbf{x}) := \frac{1}{R} \sum_{r=1}^R p(y | \mathbf{x}, \hat{\mathbf{w}}_r).$$

ВАРИАЦИОННЫЙ АВТОКОДИРОВЩИК

- (Kingma et al., 2013) Вариационный автокодировщик (variational autoencoder, VAE)
- Давайте в методе главных компонент подставим нелинейную функцию вместо линейной:

$$p(X, Z \mid \theta) = \prod_{i=1}^m p(x_i, z_i \mid \theta) = \prod_{i=1}^m p(x_i \mid z_i, \theta)p(z_i \mid \theta) = \\ \prod_{i=1}^m \prod_{j=1}^D N(x_{ij} \mid \mu_j(z_i), \sigma_j(z_i))N(z_i \mid 0, I).$$

- $\mu_j(z_i)$ и $\sigma_j(z_i)$ теперь будут нейронной сетью задаваться

ВАРИАЦИОННЫЙ АВТОКОДИРОВЩИК

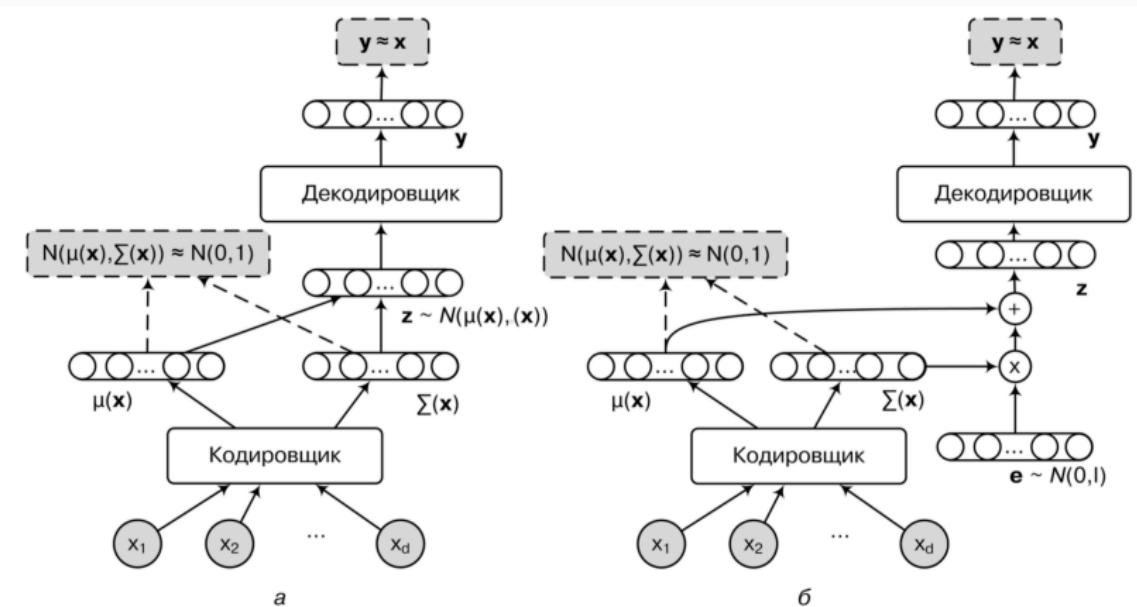
- И аппроксимировать будем тоже нейронной сетью:

$$q(Z | X, \phi) = \prod_{i=1}^N p(z_i | x_i, \phi) = \prod_{i=1}^N \prod_{j=1}^D N(z_i | \mu_j(x_i), \sigma_j(x_i)).$$

- Две сети: первая d -мерный вектор скрытых переменных \mathbf{z} превращает в вектор размерности $2D$ параметров распределений на \mathbf{x} , вторая получает D -мерный вектор \mathbf{x} и выдаёт вектор размерности $2d$ с параметрами распределений на скрытые переменные \mathbf{z} .

ВАРИАЦИОННЫЙ АВТОКОДИРОВЩИК

- Слева основная идея, справа репараметризация (потом):



ВАРИАЦИОННЫЙ АВТОКОДИРОВЩИК

- Как обучить это всё?

$$\mathcal{L}(q(Z \mid X, \phi), \theta) = \mathcal{L}(\phi, \theta) = \sum_{i=1}^n \int q(z_i \mid x_i, \phi) \log \frac{p(x_i, z_i \mid \theta)}{q(z_i \mid x_i, \phi)} dz_i,$$

- Надо уметь считать градиент или хотя бы стохастический градиент...
- Если надо

$$f(x) = \mathbb{E}_{p(y)}[h(x, y)] = \int h(x, y)p(y)dy,$$

то можно породить точку y_0 из $p(y)$ и посчитать производную

$$\int p(y) \frac{\partial h(x, y)}{\partial x} dy = \frac{\partial}{\partial x} \int p(y)h(x, y)dy = \frac{\partial f}{\partial x}.$$

ВАРИАЦИОННЫЙ АВТОКОДИРОВЩИК

- Пусть распределение в мат. ожидании зависит от x :

$$f(x) = \int h(x, y)p(y | x)dy.$$

- Дифференцируем как произведение:

$$\frac{\partial f}{\partial x} = \frac{\partial}{\partial x} \int h(x, y)p(y | x)dy = \int \left[\frac{\partial h(x, y)}{\partial x} p(y | x) + h(x, y) \frac{\partial p(y | x)}{\partial x} \right] dy.$$

- Первый интеграл можно теперь оценить как раньше.
- А вот со вторым теперь проблема: нигде нет математического ожидания. Что делать?..

ВАРИАЦИОННЫЙ АВТОКОДИРОВЩИК

- Log-derivative trick: если хочется представить $\frac{\partial p(y|x)}{\partial x}$ как $p(y|x)$, умноженное на что-то, то можно представить это в виде

$$\frac{\partial p(y|x)}{\partial x} = p(y|x) \frac{\partial \log p(y|x)}{\partial x}.$$

- Итого большой интеграл можно переписать как

$$\int p(y|x) \left[\frac{\partial h(x,y)}{\partial x} + h(x,y) \frac{\partial \log p(y|x)}{\partial x} \right] dy.$$

- Подсчитать это честно по-прежнему не получится, но достаточно получить несмещённую оценку: породим $y \sim p(y|x)$ и подставим:

$$\frac{\partial h(x,y_0)}{\partial x} + h(x,y_0) \frac{\partial \log p(y_0|x)}{\partial x}, \quad y_0 \sim p(y|x).$$

- Опять же, можно сгенерировать несколько и усреднить результаты.

ВАРИАЦИОННЫЙ АВТОКОДИРОВЩИК

- Теперь возвращаемся: нам нужно уметь $\mathcal{L}(\phi, \theta)$ максимизировать по отдельности по ϕ и по θ .
- Максимизация по θ – это простой случай стохастического градиента, при фиксированных ϕ получится просто конкретное распределение $p(z_i | x_i, \phi)$, из него возьмём выборку:

$$\begin{aligned}\frac{\partial}{\partial \theta} \mathcal{L}(\theta, \phi) &= \frac{\partial}{\partial \theta} \sum_{i=1}^n \int q(z_i | x_i, \phi) \log \frac{p(x_i, z_i | \theta)}{q(z_i | x_i, \phi)} dz_i \approx \\ &\approx n \frac{\partial}{\partial \theta} \log p(x_i, z | \theta).\end{aligned}$$

- Здесь мы взяли случайный объект x_i из выборки (равномерно) и сгенерировали для него случайный $z \sim p(z | x_i, \phi)$.

ВАРИАЦИОННЫЙ АВТОКОДИРОВЩИК

- А стохастический градиент по ϕ получается так, как описано выше, при помощи log-derivative trick.
- Важная проблема: у стохастических градиентов очень большая дисперсия, и понизить её пока в общем случае не получается.
- Здесь ещё один трюк: репараметризация (reparametrization trick): если есть выражение вида $\int f(x)p(x \mid \theta)dx$, и мы хотим его продифференцировать по θ , давайте устраним параметры из распределения заменой переменных:

$$\begin{aligned}\frac{\partial}{\partial \theta} \int f(x)p(x \mid \theta)dx &= [p(x \mid \theta)dx = p(\epsilon)d\epsilon, x = g(\epsilon, \theta)] = \\ &= \frac{\partial}{\partial \theta} \int f(g(\epsilon, \theta))p(\epsilon)d\epsilon \approx \frac{\partial}{\partial \theta} f(g(\epsilon_0, \theta)),\end{aligned}$$

Где $\epsilon_0 \sim p(\epsilon)$.

ВАРИАЦИОННЫЙ АВТОКОДИРОВЩИК

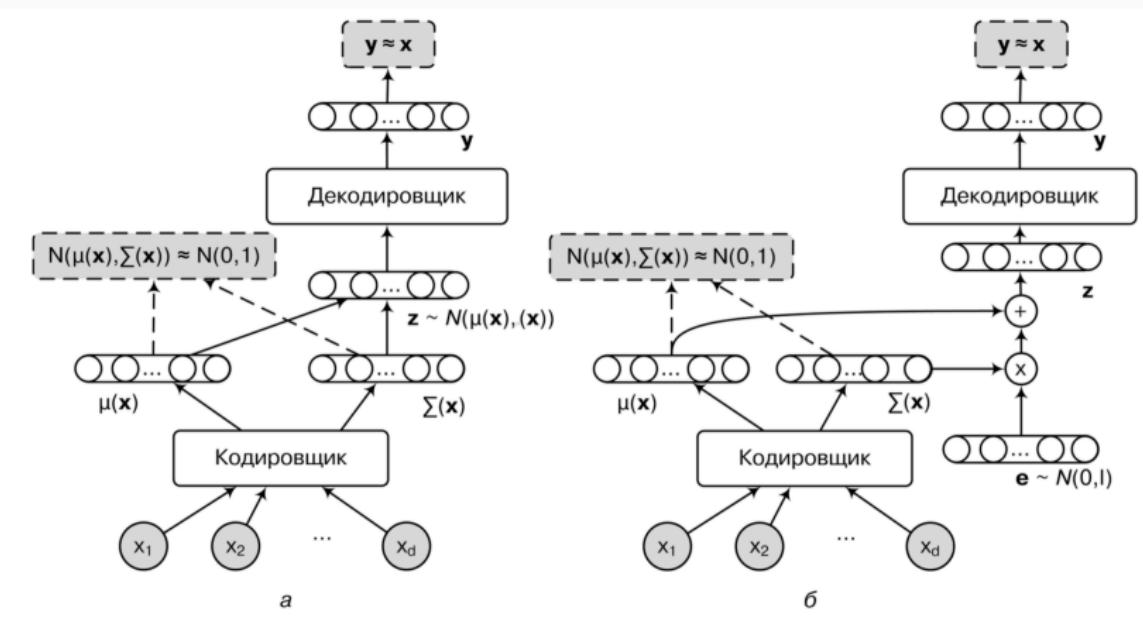
- Такой трюк возможен не всегда, но в нашем случае, когда распределения параметризованы нейросетями, он всегда проходит.
- Можно получить выражение для градиента по ϕ :

$$\begin{aligned}\frac{\partial}{\partial \phi} \mathcal{L}(\theta, \phi) &= \frac{\partial}{\partial \phi} \sum_{i=1}^n \int q(z_i | x_i, \phi) \log \frac{p(x_i, z_i | \theta)}{q(z_i | x_i, \phi)} dz_i = \\ &[q(z_i | x_i, \phi) dz_i = g(x_i, \epsilon) = N(\epsilon | 0, I) d\epsilon, z_i = \mu(x_i) + \sigma(x_i) \cdot \epsilon] \\ &= \frac{\partial}{\partial \phi} \sum_{i=1}^n n \int g(x_i, \epsilon) \log \frac{p(x_i, g(x_i, \epsilon) | \theta)}{q(g(x_i, \epsilon) | x_i, \phi)} d\epsilon \approx n \frac{\partial}{\partial \phi} \log \frac{p(x_i, g(x_i, \epsilon) | \theta)}{q(g(x_i, \epsilon) | x_i, \phi)}\end{aligned}$$

и теперь последнее выражение уже вполне можно и подсчитать, и дифференцировать, потому что это просто выходы наших двух нейронных сетей.

ВАРИАЦИОННЫЙ АВТОКОДИРОВЩИК

- Слева основная идея, справа репараметризация (теперь понятно):



VQ-VAE

- А куда пришла эта идея сегодня? Да примерно туда же, куда и GAN'ы, и ещё неизвестно, кто победит...
- VQ-VAE (Vector Quantized VAE; Razavi et al., 2019, DeepMind):



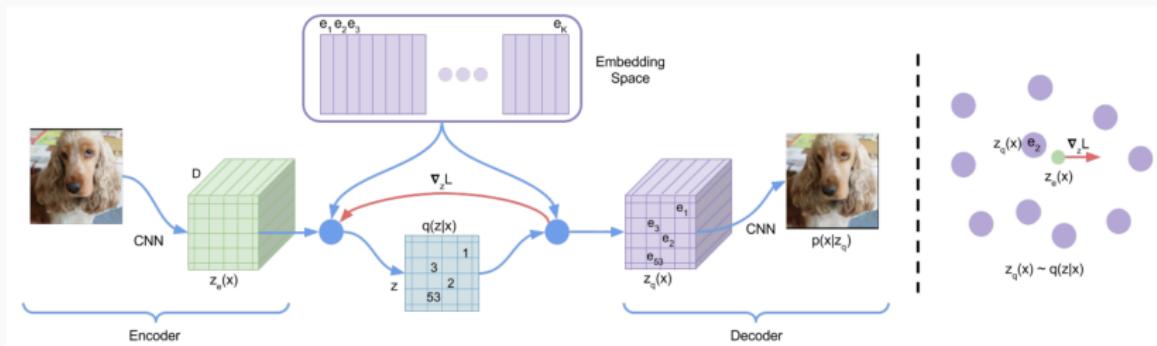
- Как такая красота получается?

VQ-VAE

- Идея VQ-VAE (van den Oord et al., 2017): давайте квантизуем латентный код, т.е. код – это как бы «язык», на котором переговариваются кодировщик и декодировщик:

$$\text{Quant}(\mathbf{z}_e(\mathbf{x})) = \mathbf{e}_k, \quad k = \arg \min_j \|\mathbf{z}_e(\mathbf{x}) - \mathbf{e}_j\|_2.$$

- Через это градиент не пропустишь, но давайте градиент просто скопируем:



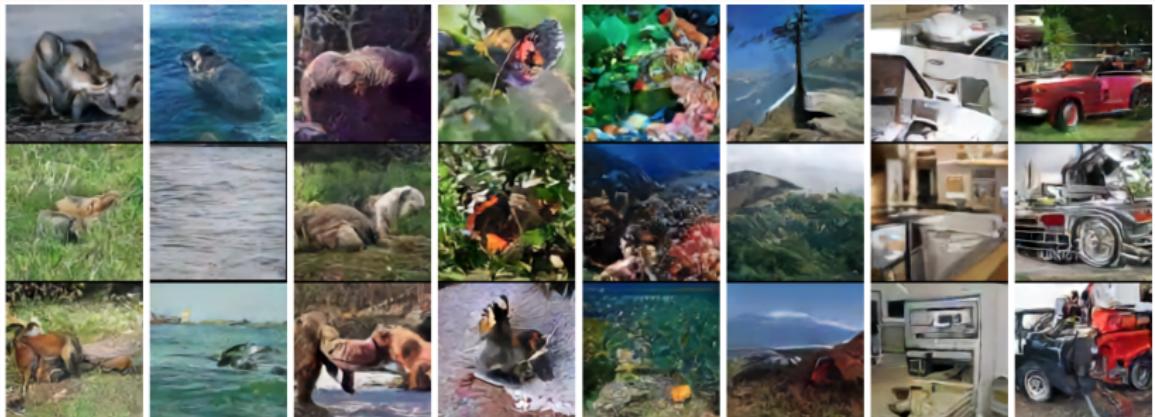
- А как обучать тогда эти embeddings, если там нет градиентов от реконструкции?
- Давайте просто добавим обычным vector quantization алгоритмом, т.е. будем подтягивать \mathbf{e} к выходам кодировщика $\mathbf{z}_e(\mathbf{x})$.
- А чтобы кодировщик не убегал неограниченно от embeddings, давайте его тоже к этим embeddings подтягивать.
- Т.е. функция ошибки получается такая:

$$L = \log p(\mathbf{x} | \mathbf{z}_q(\mathbf{x})) + \|\text{sg}[\mathbf{z}_e(\mathbf{x})] - \mathbf{e}\|_2^2 + \beta \|\mathbf{z}_e(\mathbf{x}) - \text{sg}[\mathbf{e}]\|_2^2,$$

где sg (stopgradient) – это оператор, который останавливает градиент (1 в прямую сторону, 0 в обратную); т.е. декодер оптимизирует первое слагаемое, кодировщик первое и третье, а сами embeddings оптимизируются вторым слагаемым.

VQ-VAE

- Уже исходный VQ-VAE на основе PixelCNN неплохо выглядел для 2017 года:



VQ-VAE

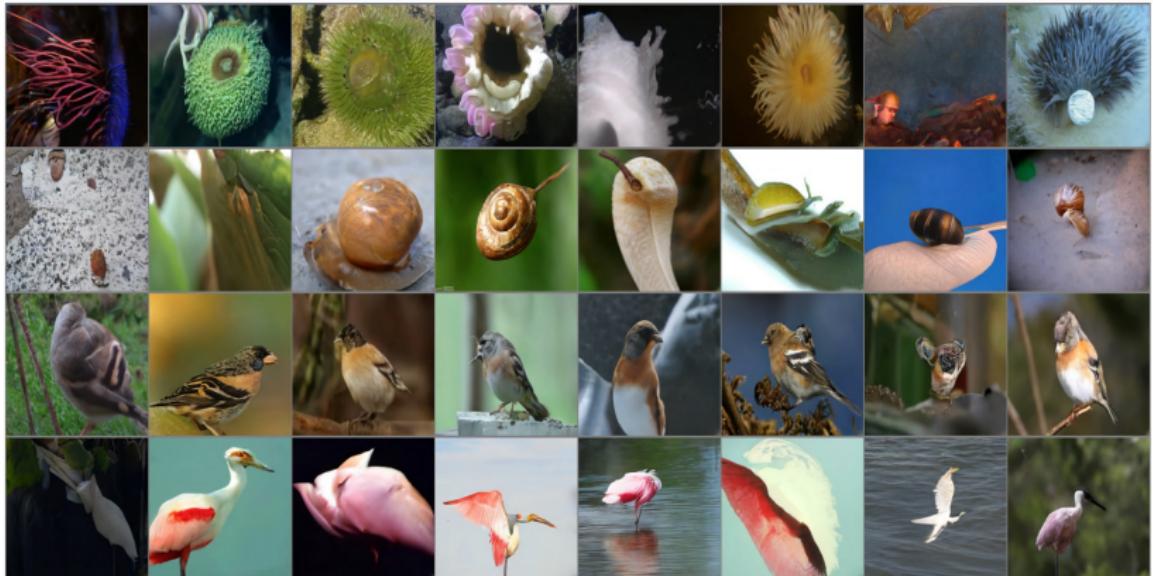
- А в VQ-VAE-2 процесс сделали иерархическим:



- Пример реконструкций по разным латентным картам:



- Получается круто:



- Но NVIDIA и тут впереди: NVAE (Nouveau VAE), (Vahdat, Kautz, 2020)
- Пожалуй, лучшее порождение на данный момент:



- Идея та же, иерархическая:

$$p(\mathbf{z}) = \prod_l p(\mathbf{z}_l \mid \mathbf{z}_{<l}),$$

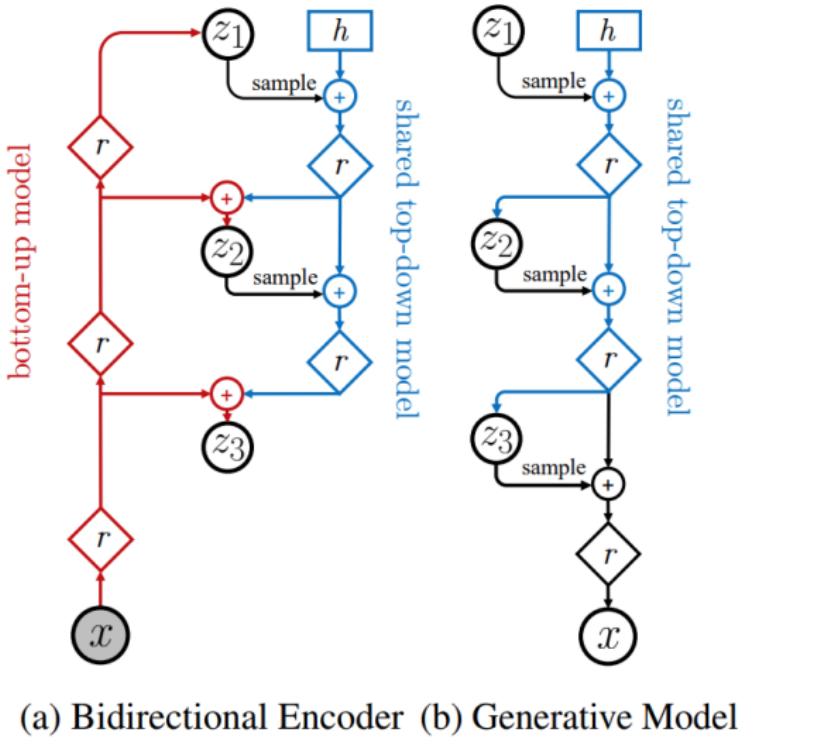
$$q(\mathbf{z} \mid \mathbf{x}) = \prod_l q(\mathbf{z}_l \mid \mathbf{z}_{<l}, \mathbf{x}),$$

$$\begin{aligned} L_{\text{VAE}}(\mathbf{x}) &= \mathbb{E}_q [\log p(\mathbf{x} \mid \mathbf{z})] - \text{KL}(q(\mathbf{z}_1 \mid \mathbf{x}) \| p(\mathbf{z}_1)) \\ &\quad - \sum_{l=2}^L \mathbb{E}_{q(\mathbf{z}_{<l} \mid \mathbf{x})} [\text{KL}(q(\mathbf{z}_l \mid \mathbf{x}, \mathbf{z}_{<l}) \| p(\mathbf{z}_l \mid \mathbf{z}_{<l}))], \end{aligned}$$

где $q(\mathbf{z}_{<l} \mid \mathbf{x}) = \prod_{i=1}^{l-1} q(\mathbf{z}_i \mid \mathbf{x}, \mathbf{z}_{<i})$.

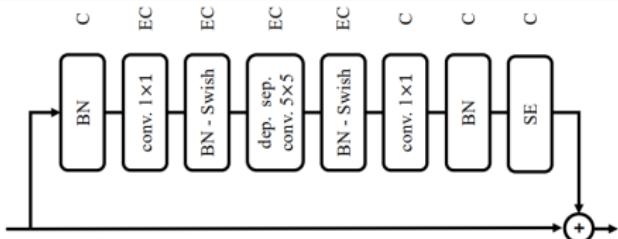
- Обучается как в VAE, через reparametrization trick.

- Вот так это выглядит:

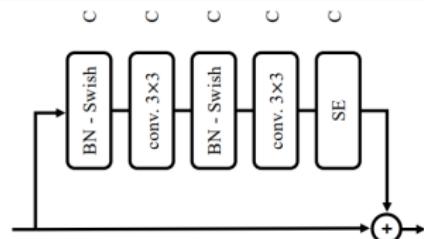


NVAE

- В NVAE постарались сделать хорошие архитектуры:



(a) Residual Cell for NVAE Generative Model



(b) Residual Cell for NVAE Encoder

- И трюки для стабилизации обучения (иерархически нелегко это сделать) и для экономии памяти.
- В том числе, кстати, нормализующие потоки используют, к которым мы потом перейдём.

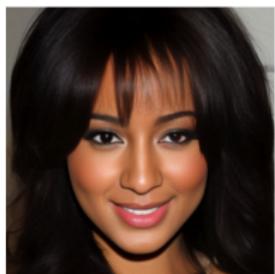
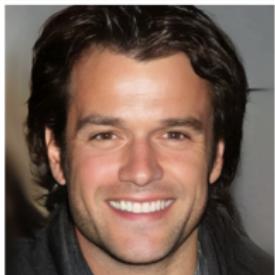
- В итоге получаются очень крутые реконструкции:



- Но это не запоминание:



- А вот ещё красивые сэмплы:



SUPERRESOLUTION

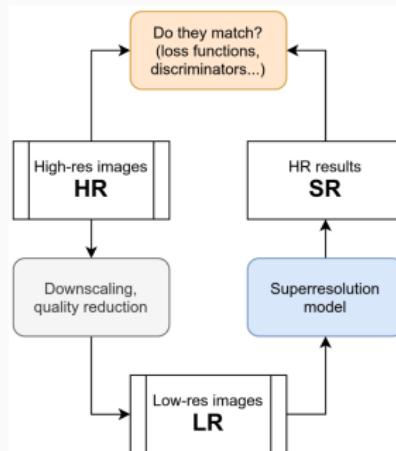
SUPERRESOLUTION

- Смысл задачи – как перейти от картинок слева к картинкам справа?
- Т.е. надо научиться увеличивать разрешение фотографий



SUPERRESOLUTION

- Почти все решения работают так:
 - берём датасет картинок высокого разрешения
 - делаем из них картинки низкого разрешения
 - обучаем модель восстанавливать high-res; т.е. paired dataset как бы и не нужен...



- Прежде чем начнём что-то делать: как мы будем измерять результаты?
- Стандартные метрики:
 - SSIM, structural similarity index: для двух окон x и y

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)},$$

где $c_1 = (0.01 \cdot L)^2$, $c_2 = (0.03 \cdot L)^2$, L – максимальное значение пикселя (динамический диапазон)

- это на самом деле комбинация трёх метрик: luminance, contrast, structure
- PSNR, peak signal-to-noise ratio – это просто функция от MSE:

$$\text{MSE}(x, y) = \frac{1}{mn} \sum_{i,j} (x_{i,j} - y_{i,j})^2,$$

$$\text{PSNR}(x, y) = 10 \log_{10} \left(\frac{\max_x}{\text{MSE}} \right),$$

где \max_x – максимальное значение пикселя

SUPERRESOLUTION

- SSIM и PSNR – основные метрики в литературе по SR, да и вообще в оценках качества, например, компрессии изображений
- Но в какой-то момент они перестают уже различать модели, и они не точно соответствуют человеческому взгляду
- Поэтому есть большая область разработки и сравнения разных метрик качества; (Athar, Wang, 2019):

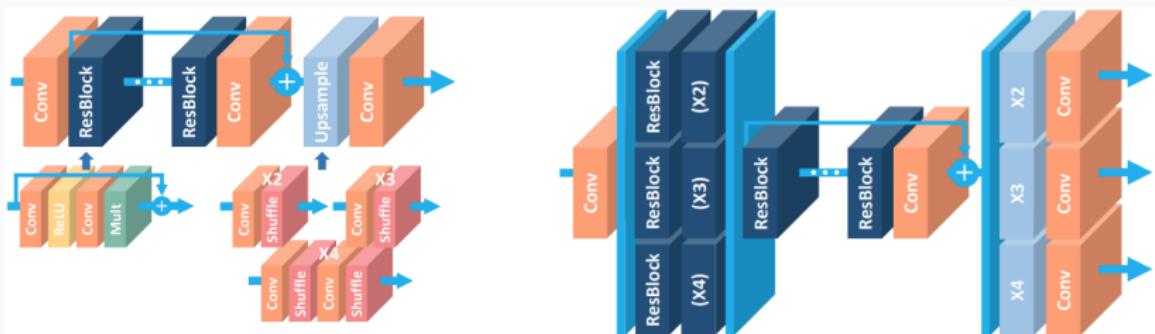
| Part I: All Databases | | | |
|-----------------------|--------|-----------|--------|
| FR Method | PLCC | FR Method | SRCC |
| RAS5 | 0.8985 | RAS4 | 0.8907 |
| RAS6 | 0.8979 | RAS5 | 0.8903 |
| RAS4 | 0.8977 | RAS1 | 0.8783 |
| RAS_MMF4 | 0.8967 | RAS2 | 0.8777 |
| RAS7 | 0.8935 | RAS_MMF4 | 0.8771 |
| RAS_MMF3 | 0.8912 | RAS6 | 0.8761 |
| RAS3 | 0.8911 | RAS_MMF3 | 0.8724 |
| RAS1 | 0.8908 | RAS7 | 0.8710 |
| RAS_MMF2 | 0.8888 | RAS_MMF2 | 0.8690 |
| RAS_B1 | 0.8881 | RAS3 | 0.8665 |
| RAS2 | 0.8879 | RAS_B1 | 0.8653 |
| RAS_B2 | 0.8850 | CISI | 0.8634 |
| CISI | 0.8831 | VSI | 0.8631 |
| IWSSIM | 0.8787 | FSIMc | 0.8628 |
| RAS_MMF1 | 0.8786 | RAS_B2 | 0.8607 |
| ESIM | 0.8705 | IWCCIM | 0.8550 |

- Например:
 - VSI, Visual Saliency-Induced Index (Zhang et al., 2014): сравниваем, взвешивая пиксели по saliency maps, которые пытаются найти патчи, важные для человеческого внимания;
 - SFF, Sparse Feature Fidelity (Chang et al., 2013): сравниваем, переводя сначала в разреженные представления, которые должны, по идее, соответствовать тому, что делает зрительная кора...
- В общем, большая наука, далёкая от завершения. Давайте вернёмся к собственно методам superresolution.

- Есть три основных подхода, на которые можно классифицировать методы. Оказывается, что от метода, которым мы пользуемся для уменьшения картинок, очень многое зависит:
 - *non-blind SR* – зафиксируем некоторое «идеальное» ядро для уменьшения (например, бикубическое); тогда всё будет работать очень хорошо, когда ядро действительно совпадает, но будет быстро ухудшаться, когда ядро будет другим, а оно ведь будет другим (Shocher et al., 2018; Lim et al., 2017; Zhang et al., 2018)
 - *blind SR* – методы, которые стараются не делать предположений о ядре
 - *kernel estimation* – давайте заставим наш *blind SR* «прозреть», обучив подходящее ядро для каждой отдельной картинки.

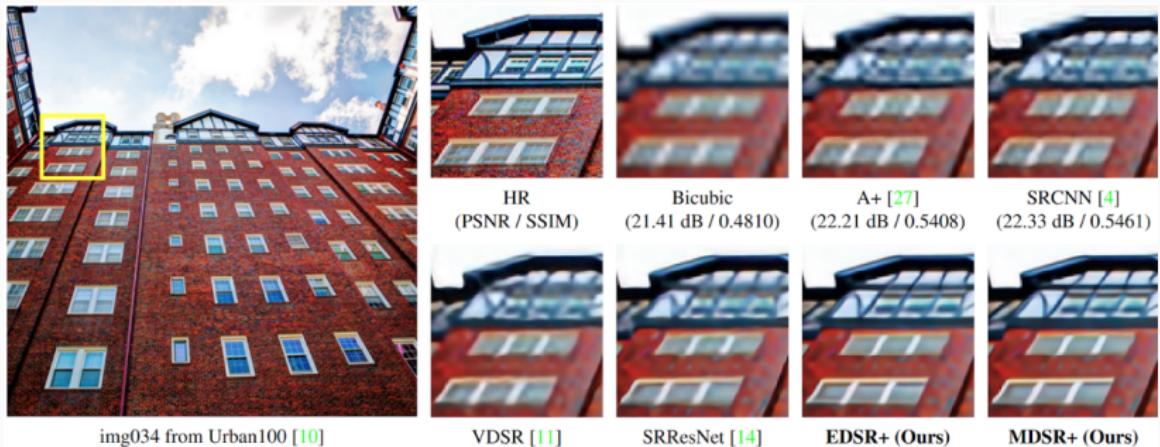
SUPERRESOLUTION

- Пример supervised подхода: EDSR (Enhanced Deep Super-Resolution network), уже классика (Lim et al., 2017)
- Обучаем просто свёрточную сеть восстанавливать картинки; есть single-scale и multi-scale варианты



SUPERRESOLUTION

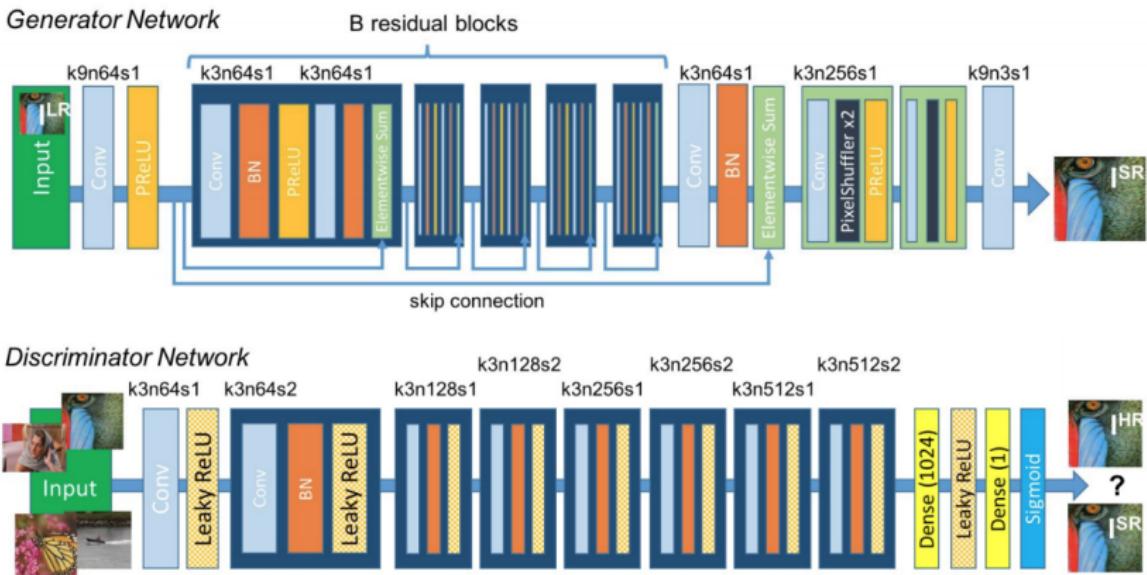
- EDSR долго определял state of the art и вообще не так уж плохо работал:



- А дальше начались как раз GAN'ы...

SUPERRESOLUTION

- SRGAN (Ledig et al., 2016): базовая схема GAN-based SR:
 - генератор делает SR
 - дискриминатор пытается отличить результаты генератора от настоящих HR картинок



- Дискриминатор оптимизирует обычную кросс-энтропию, как всегда:

$$\max_D \mathbb{E}_{I_{\text{HR}}} [\log D(I_{\text{HR}})] + \mathbb{E}_{I_{\text{LR}}} [\log(1 - D(G(I_{\text{LR}})))]$$

- А интересная часть – как сделать функцию ошибки для генератора:

$$L^{\text{SR}} = L_*^{\text{SR}} + \lambda \cdot L_{\text{adv}}^{\text{SR}},$$

где $L_{\text{adv}}^{\text{SR}}$ – adversarial loss, как обычно:

$$L_{\text{adv}}^{\text{SR}} = \sum_{n=1}^N -\log D(G(I_{\text{LR}}))$$

SUPERRESOLUTION

- А вот L_*^{SR} – content loss – уже интереснее; SRGAN рассматривают MSE как вариант, но пишут, что лучше работает VGG loss:

$$L_{\text{VGG}}^{\text{SR}} = \frac{1}{WH} \sum_{i=1}^w \sum_{j=1}^h (\phi(I_{\text{HR}})_{ij} - \phi(G(I_{\text{LR}}))_{ij})^2,$$

где ϕ – признаки, выделенные где-то в VGG-сети

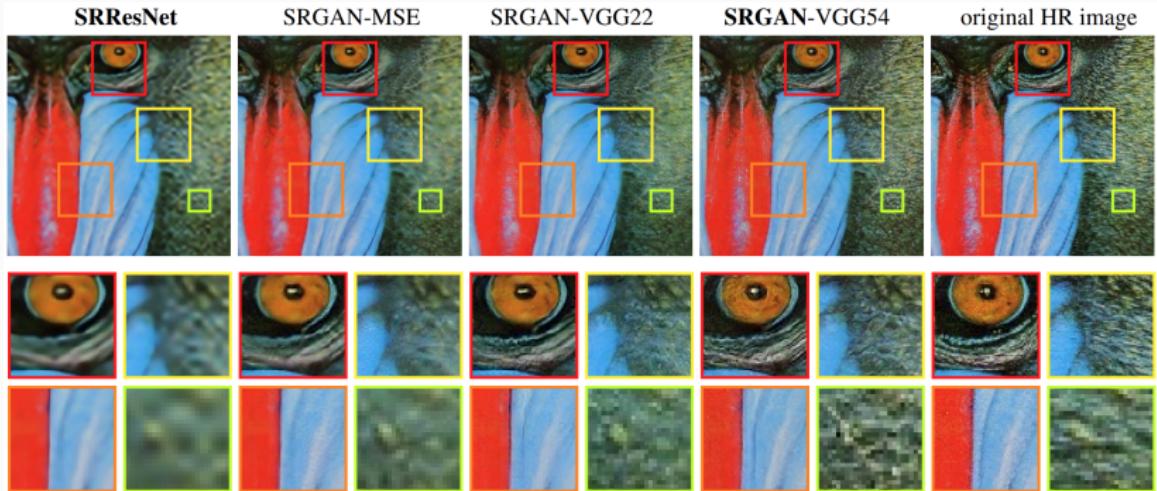
- Разница прямо есть:

| Set5 | SRResNet- | | SRGAN- | | |
|-------------|-----------|--------|--------|--------|--------|
| | 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 | | | | | |
|--------------|--------|--------|--------|--------|--------|
| | MSE | VGG22 | MSE | VGG22 | VGG54 |
| 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* |

SUPERRESOLUTION

- И на практике тоже видна эта разница:



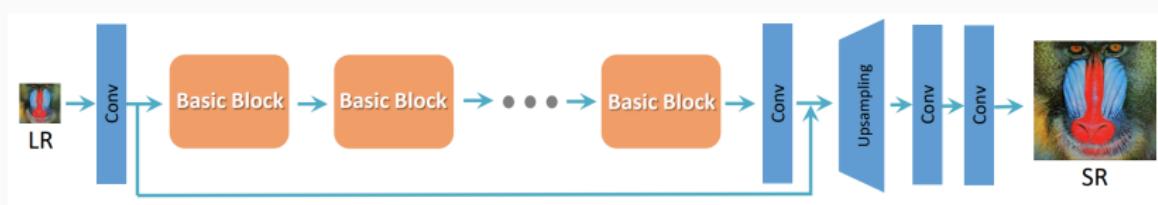
SUPERRESOLUTION

- Кажется, что уже лучше трудно и придумать, но на самом деле всё только начинается...

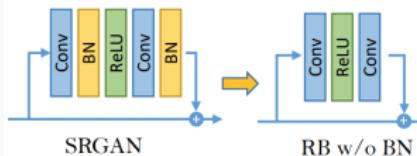


SUPERRESOLUTION

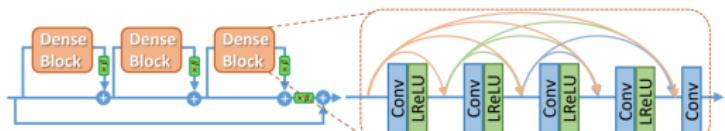
- ERSGAN (Wang et al., 2018) улучшает базовую идею SRGAN; тот же принцип, но архитектуры другие, и генератор в основном работает в feature space LR-картинок:



Residual Block (RB)



Residual in Residual Dense Block (RRDB)



SUPERRESOLUTION

- Relativistic discriminator:

$$D(x_r) = \sigma(C(\text{Real})) \rightarrow 1 \quad \text{Real?}$$

$$D(x_f) = \sigma(C(\text{Fake})) \rightarrow 0 \quad \text{Fake?}$$



a) Standard GAN

$$D_{Ra}(x_r, x_f) = \sigma(C(\text{Real}) - \mathbb{E}[C(\text{Fake})]) \rightarrow 1$$

$$D_{Ra}(x_f, x_r) = \sigma(C(\text{Fake}) - \mathbb{E}[C(\text{Real})]) \rightarrow 0$$

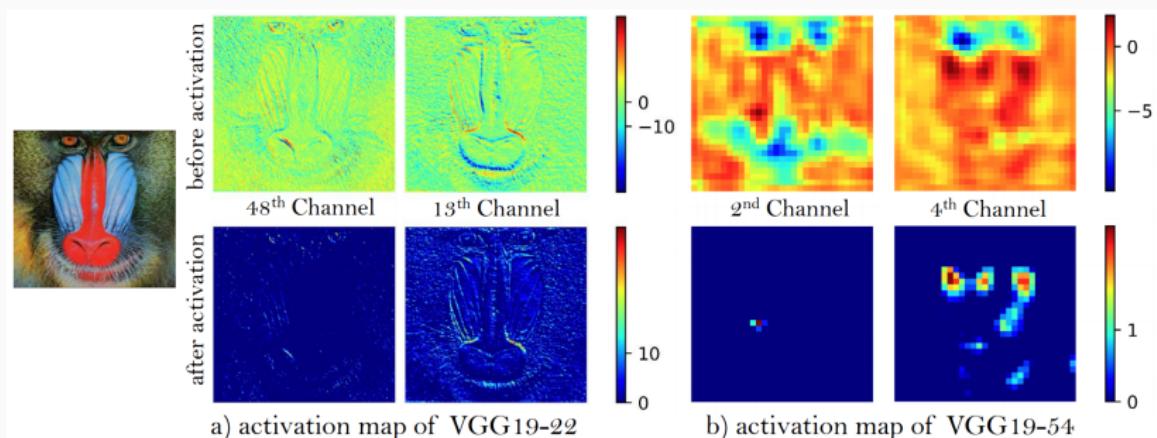
b) Relativistic GAN

More realistic
than fake data?

Less realistic
than real data?

SUPERRESOLUTION

- Perceptual loss тоже немножко изменился – теперь берём признаки не после функции активации, а до, там меньше разреженность:



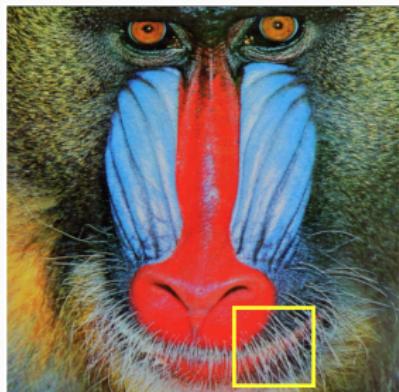
- Но в принципе то же самое:

$$L^{\text{SR}} = L_*^{\text{SR}} + \lambda \cdot L_{\text{adv}}^{\text{SR}} + \eta L_1.$$

- Ещё интересно, что обучают G_{PSNR} на основе PSNR, потом делают fine-tuning в виде GAN до G_{GAN} , а потом интерполируют веса между этими двумя сетями.

SUPERRESOLUTION

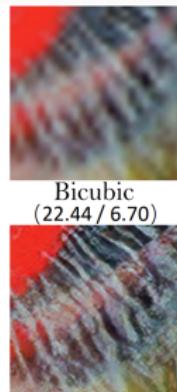
- И действительно разницу можно увидеть:



baboon from Set14
(PSNR / Perceptual Index)



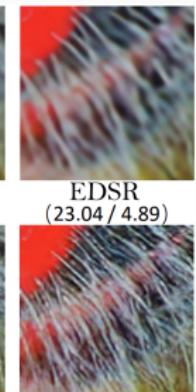
HR
(∞ / 3.59)



Bicubic
(22.44 / 6.70)



SRCCN
(22.73 / 5.73)



EDSR
(23.04 / 4.89)

RCAN
(23.12 / 4.20)

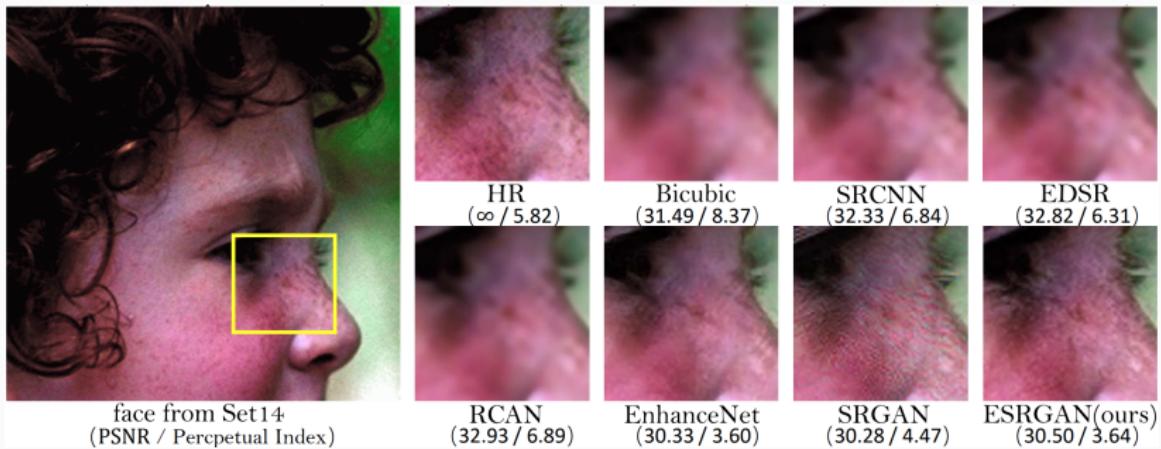
EnhanceNet
(20.87 / 2.68)

SRGAN
(21.15 / 2.62)

ESRGAN(ours)
(20.35 / 1.98)

SUPERRESOLUTION

- И действительно разницу можно увидеть:



SUPERRESOLUTION

- А вот разница между G_{PSNR} и G_{GAN} :



SUPERRESOLUTION

- ESRGAN довольно актуален до сих пор. Но есть и новые веяния
- FastAI: Decrappify, Deoldify, and Superresolution



'Migrant Mother' by Dorothea Lange (1936) colorized by DeOldify (right) and baseline algorithm (center)

SUPERRESOLUTION

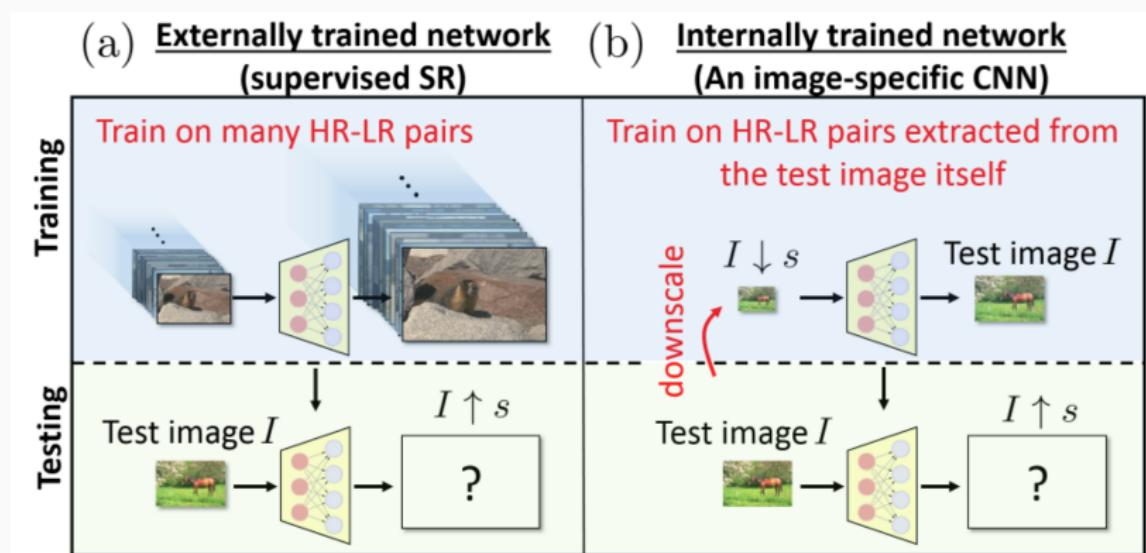
- Современная версия очень простого метода: давайте максимально ухудшим хорошие картинки, добавим кучу шума и аугментаций, и попросим восстановить
- Пишут, что пытались добавить GAN'ы, но оказалось, что в этом пайплайне GAN'ы не особенно и нужны; NoGAN approach
- <https://www.fast.ai/2019/05/03/decrappify/>



'Gypsy Camp, Maryland' (1925)

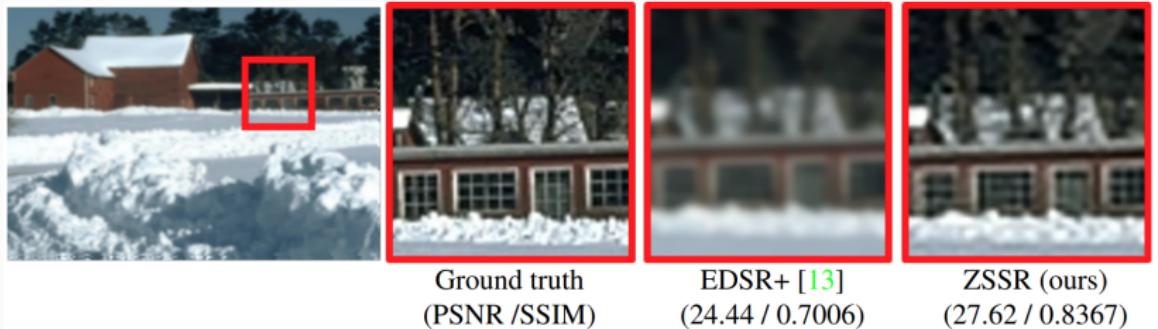
SUPERRESOLUTION

- Но всё это было в том или ином смысле non-blind. А как понять, какое ядро лучше всего восстанавливать?
- ZSSR (Zero-Shot Super-Resolution), Shocher et al. (2019): обучаем ядро прямо на текущей картинке, self-supervised



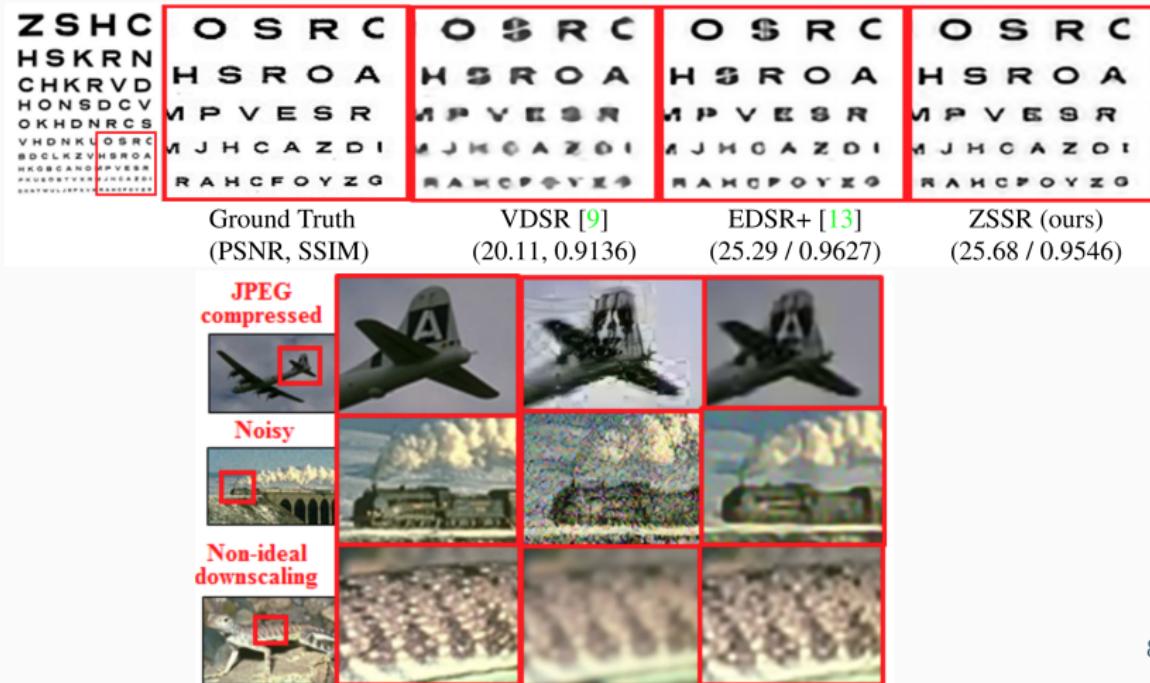
SUPERRESOLUTION

- Получается лучше в ситуациях, когда ядро неизвестно



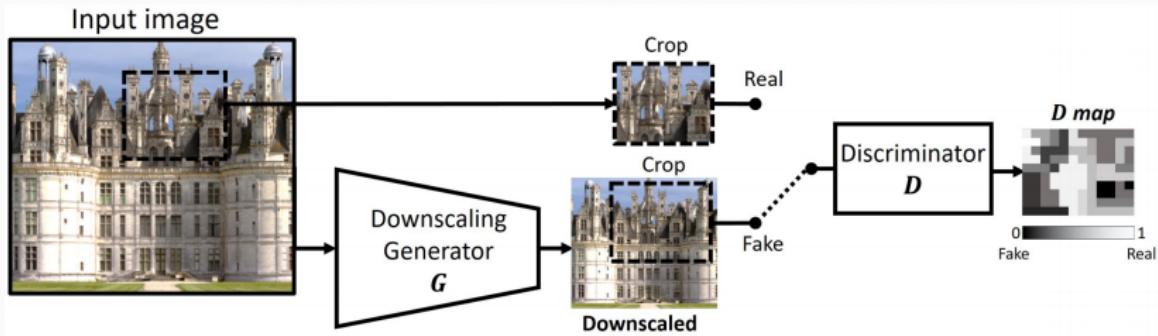
SUPERRESOLUTION

- И даже в идеальных ситуациях (с известным ядром) иногда может быть лучше, если есть повторяющиеся структуры, которые можно обучить:



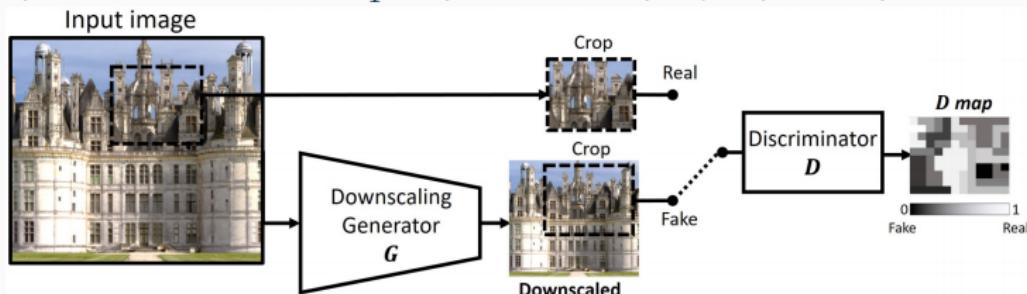
SUPERRESOLUTION

- А следующее развитие – это добавить сюда GAN
- KernelGAN (Bell-Kligler et al., 2019) обучает генератор делать downscaling так, чтобы кусочки картинки были неотличимы от исходных, а потом применяет это ядро фактически через ZSSR:

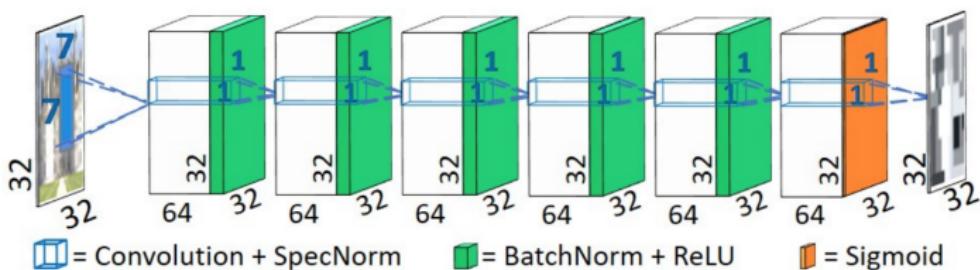


SUPERRESOLUTION

- Сети на самом деле довольно простые и маленькие: линейный генератор (чтобы был эквивалентен свёртке с ядром), LSGAN loss с L_1 -нормой плюс регуляризатор Ω

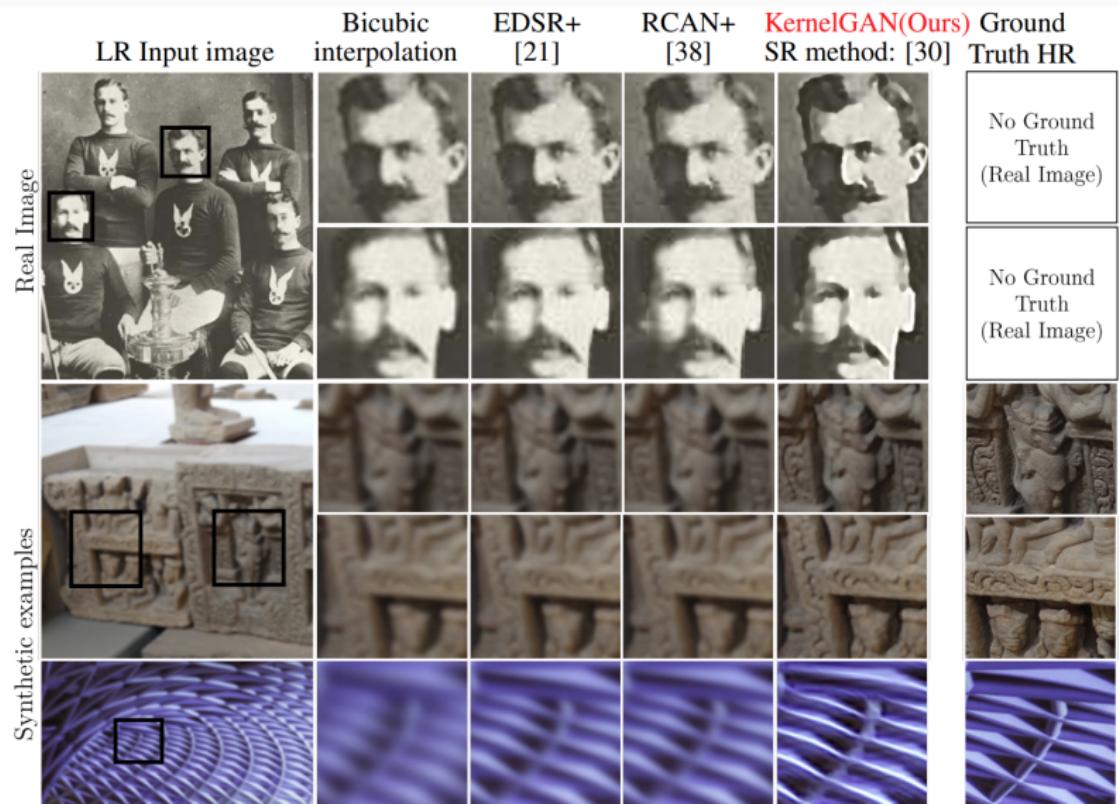


- Простой свёрточный дискриминатор:



SUPERRESOLUTION

- Получается хорошо:

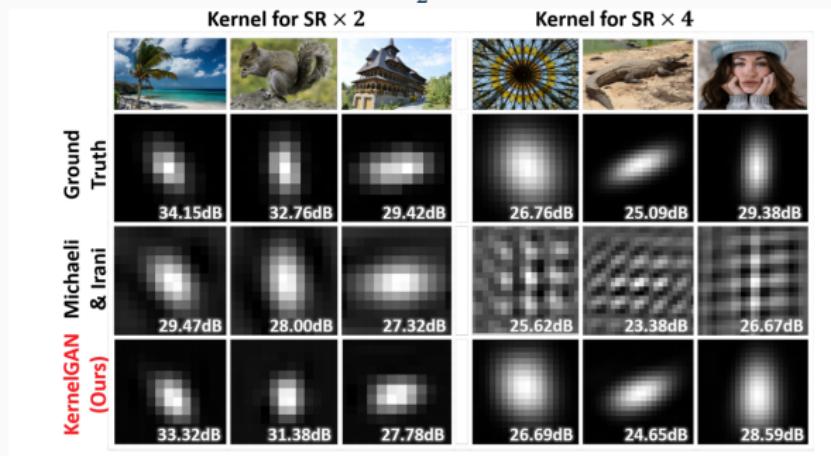


SUPERRESOLUTION

- Выделять ядро явно, кстати, полезно потому, что можно его регуляризовать; в KernelGAN:

$$\Omega = \alpha L_{\text{sum-to-1}} + \beta L_{\text{boundaries}} + \gamma L_{\text{sparse}} + \delta L_{\text{center}}, \text{ где}$$

- $L_{\text{sum-to-1}} = \left| 1 - \sum_{ij} k_{ij} \right|$
- $L_{\text{boundaries}} = \sum_{ij} |k_{ij} \cdot m_{ij}|$, где m растёт от центра к краю;
- $L_{\text{sparse}} = \sum_{ij} |k_{ij}|^{1/2}$
- $L_{\text{center}} = \left\| (x_0, y_0) - \frac{\sum_{ij} k_{ij} \cdot (i, j)}{\sum_{ij} k_{ij}} \right\|_2$

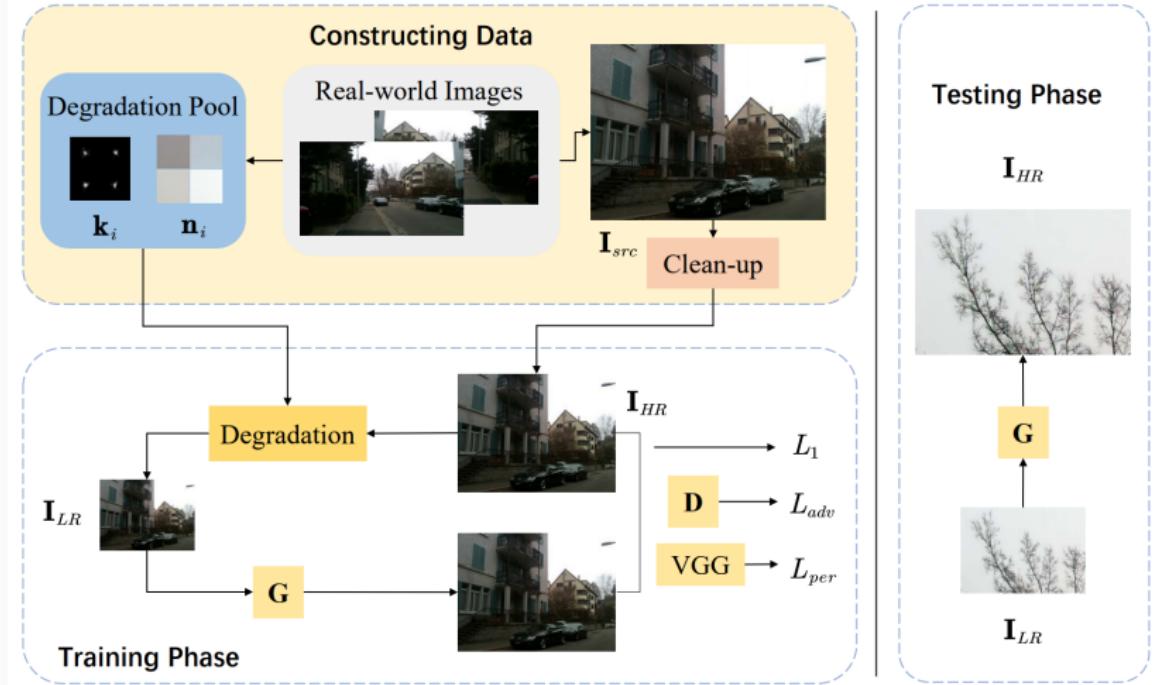


- KernelGAN хороший:
 - unsupervised
 - обучает хорошие ядра, причём для каждой картинки свои
 - давал наилучшие результаты на тот момент, когда был предложен
- Но и плохой:
 - очень медленный – надо, чтобы целое обучение сошлось для каждой картинки отдельно

- RealSR (Real-World Super-Resolution) – Ji et al., CVPR 2020:
 - идея в том, чтобы взять лучшее от обоих миров;
 - supervised SR отлично работает на идеальных датасетах, но на реальных картинках не очень, потому что ядро неизвестно;
 - KernelGAN обучает ядра, но это очень медленно и по отдельности на каждой картинке
 - так давайте обучим единую модель, но возьмём в неё много разных ядер!

SUPERRESOLUTION

- Общая идея:



- Т.е. деградация происходит при помощи ядра и шума:

$$I_{\text{LR}} = (I_{\text{HR}} * \mathbf{k}) \downarrow_s + \mathbf{n}$$

- Архитектуры по сути из ESRGAN, функция ошибки та же самая:

$$L = \lambda_1 L_1 + \lambda_{\text{perc}} L_{\text{perc}} + \lambda_{\text{adv}} L_{\text{adv}}$$

- Новое здесь только noise injection – собираем патчи шума, добавляем их как элемент деградации.

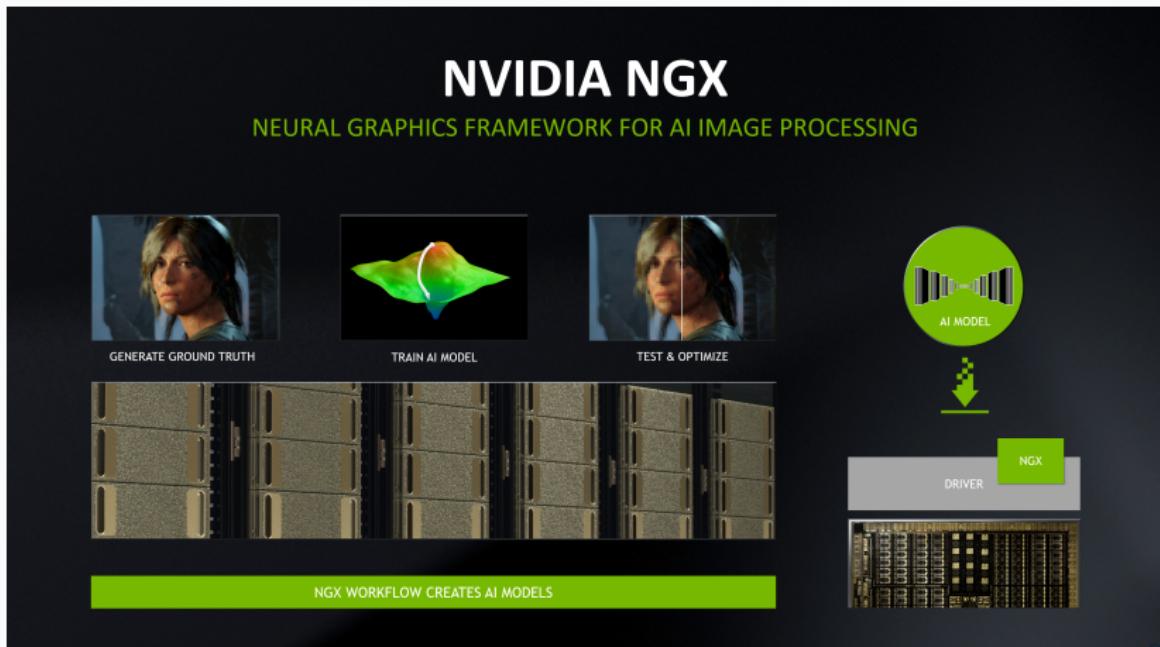
SUPERRESOLUTION

- Но получается прямо хорошо:



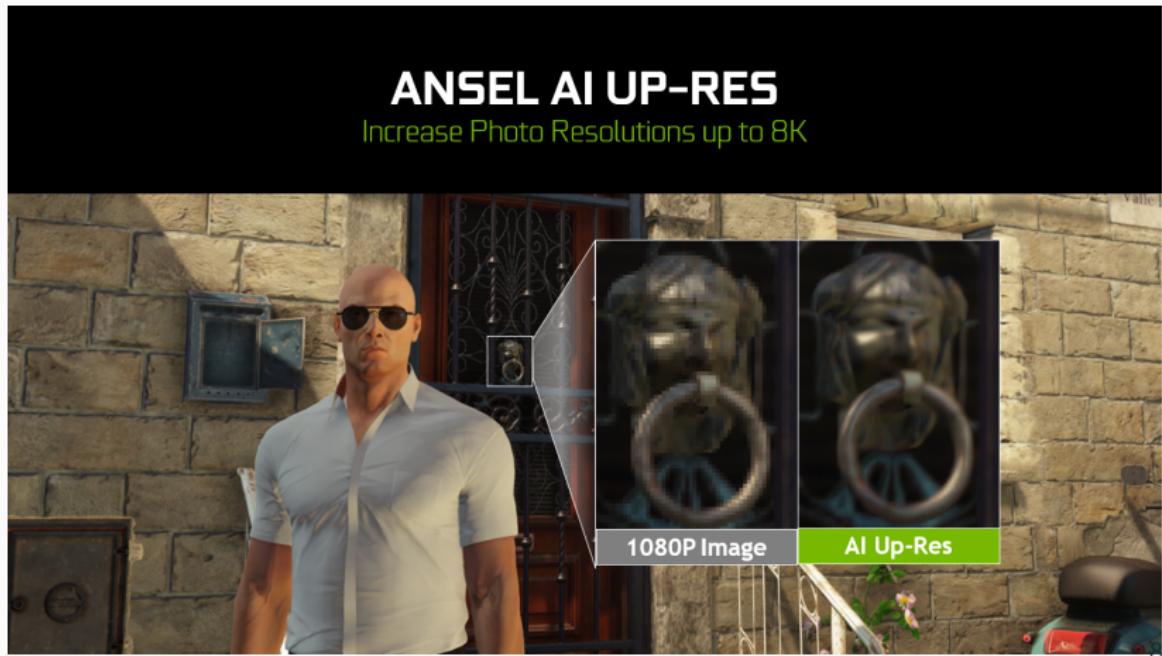
SUPERRESOLUTION

- NVIDIA DLSS (Deep Learning Super Sampling) – это не только и не столько superresolution, но интересное применение всего этого дела; первая версия вышла в 2018



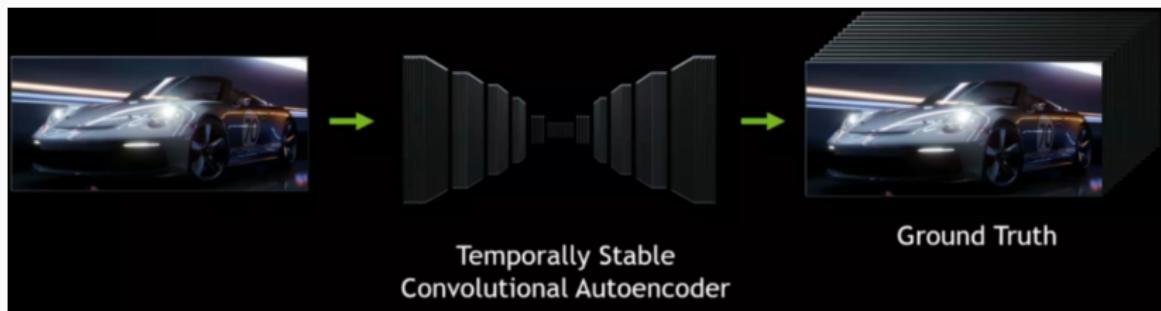
SUPERRESOLUTION

- Ещё раньше началось с NVIDIA Ansel, который делал крутые скриншоты в высоком разрешении; там уже был superresolution



SUPERRESOLUTION

- Первая версия DLSS – это свёрточный автокодировщик, обученный восстанавливать идеальные кадры, отрендеренные в высоком разрешении
- И, конечно, это non-blind SR, тут нет никакой камеры и можно обучать в реальных условиях



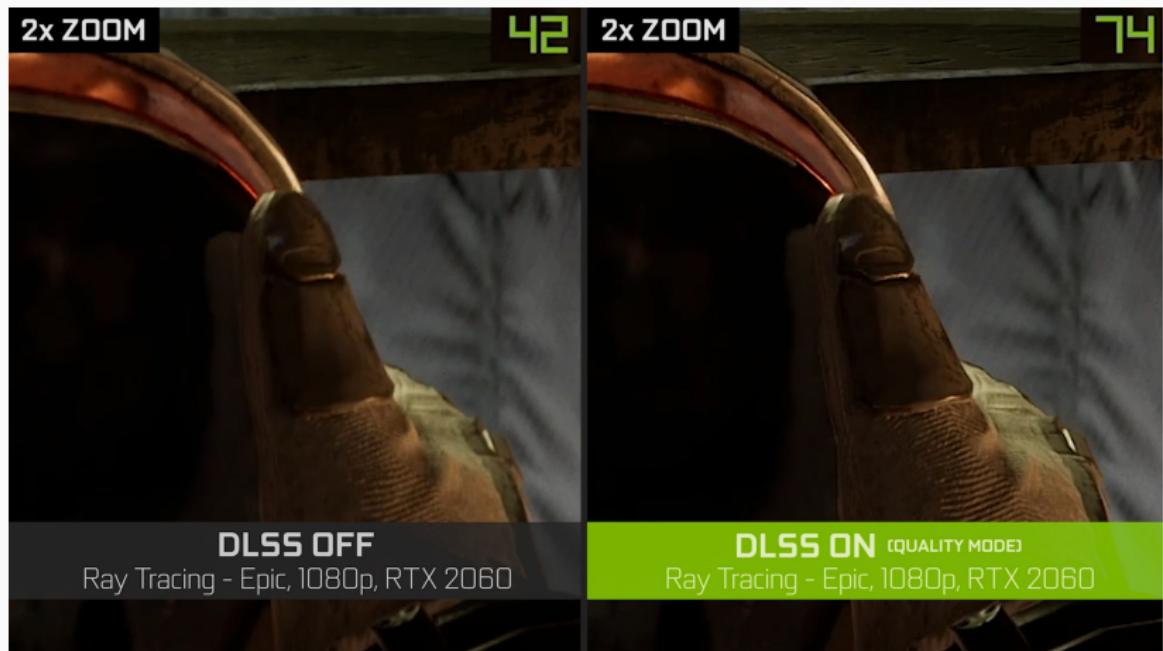
SUPERRESOLUTION

- Получалось хорошо, но обратите внимание, что DLSS в каждой игре вводился отдельно, т.е. модель нужно было обучать отдельно в каждом случае
- Так что это каждый раз была новость: Battlefield V, Metro Exodus, Final Fantasy XV...



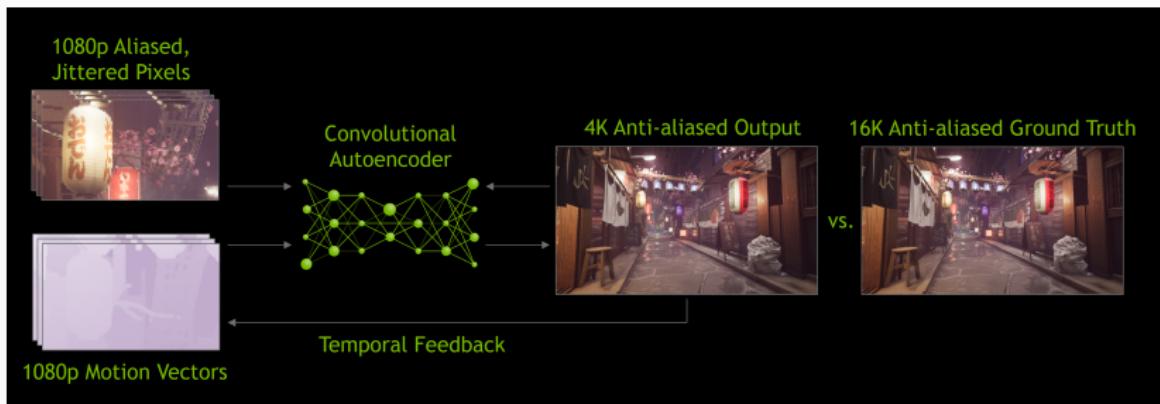
SUPERRESOLUTION

- Апрель 2020 – DLSS v 2.0



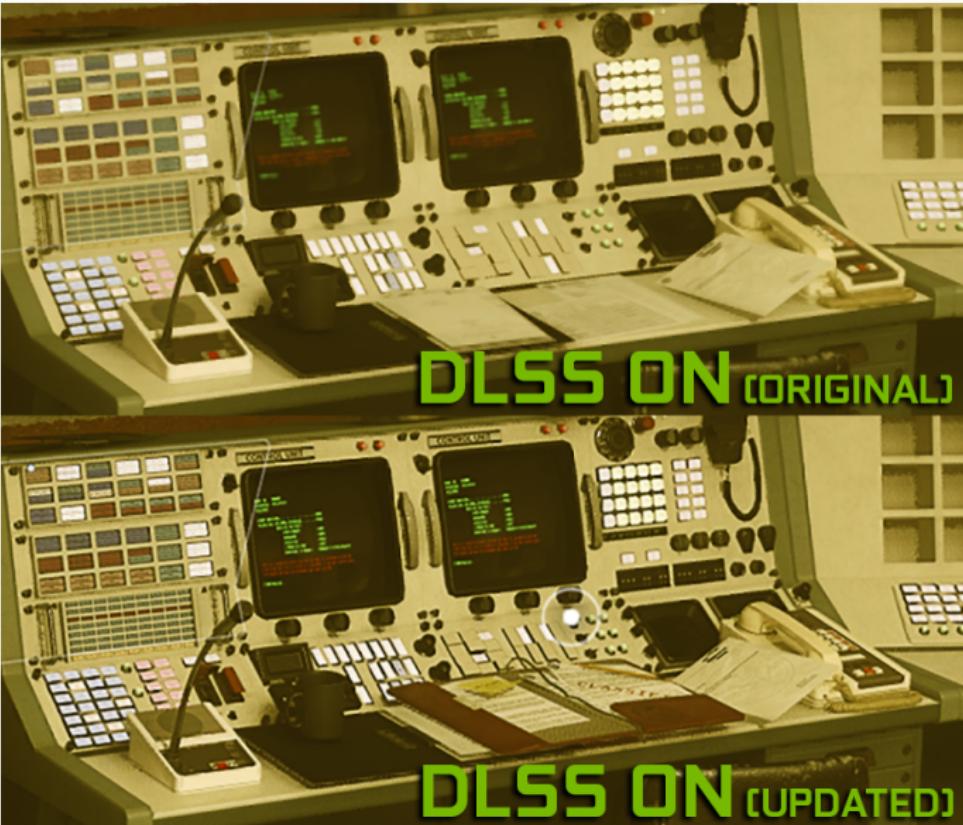
SUPERRESOLUTION

- Игровые движки могут дать не только картинку, но и motion vectors, т.е. то, куда двигаются сейчас объекты, и это тоже может помочь улучшить картинку



SUPERRESOLUTION

- Получается ещё лучше:



Спасибо!

Спасибо за внимание!

