СЕГМЕНТАЦИЯ И РЕКУРРЕНТНЫЕ СЕТИ

Сергей Николенко

СПбГУ — Санкт-Петербург

2 октября 2026 г.





Random facts:

- 2 октября 1187 г. Саладин после недолгой осады захватил Иерусалим, что послужило поводом к началу Третьего крестового похода
- 2 октября 1608 г. голландский очковый мастер Ханс Липперсгей подал заявку на патент на подзорную трубу; Галилей направил такую в небо только через год
- 2 октября 1766 г. вспыхнул ноттингемский сырный бунт: местные попытались помешать торговцам из Линкольншира вывезти сыр, купленный ими на ярмарке в Ноттингемшире, и пришлось привлекать правительственные войска
- 2 октября 1836 г. Чарльз Дарвин вернулся из длившегося почти пять лет кругосветного путешествия на корабле «Бигль»
- 2 октября 1925 г. Джон Лоджи Бэрд впервые передал по телевидению движущееся изображение
- 2 октября 1950 г. вышел первый выпуск комикса Peanuts

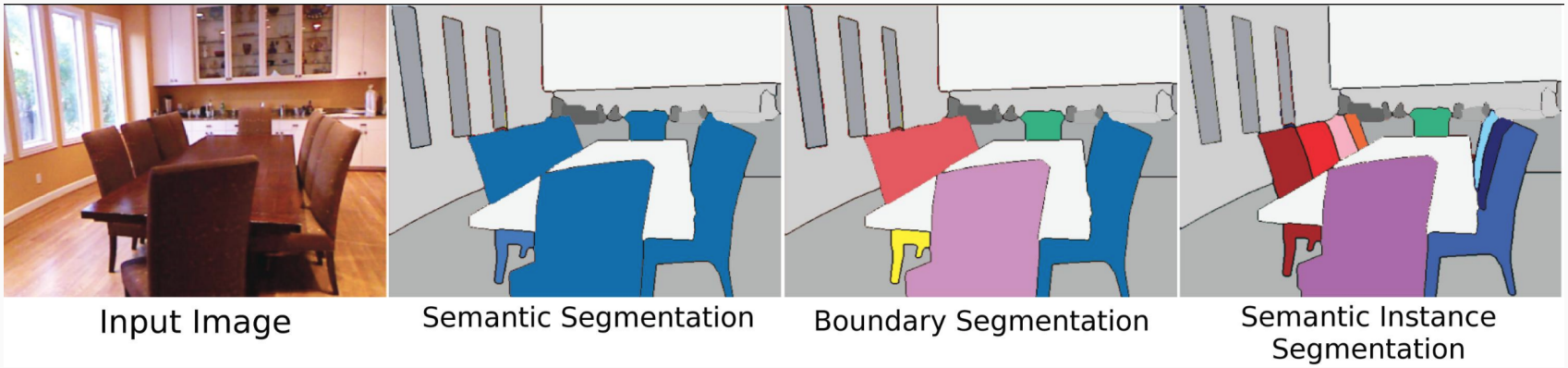
Краткое резюме предыдущей серии

- В прошлый раз мы прошли распознавание объектов целиком: двухэтапные детекторы от R-CNN до R-FCN, одноэтапные YOLO и SSD, разреженные свёртки, пирамиды признаков — и посмотрели, что стало с YOLO к 2026 году.
- Всюду там ответом был прямоугольник. Сегодня поднимем требования до пикселя: *сегментация* — какому объекту принадлежит каждая точка картинки.
- Потом — про то, как всю эту красоту ломает почти незаметный шум: *adversarial examples*.
- А во второй половине уйдём от картинок к последовательностям: *рекуррентные сети*, LSTM и GRU.

Сегментация

Сегментация: классические методы

- Сегментация: что если нам нужно знать не только где в целом расположены объекты (bounding boxes), но и более конкретно пиксельные маски?
- Разные виды сегментации:

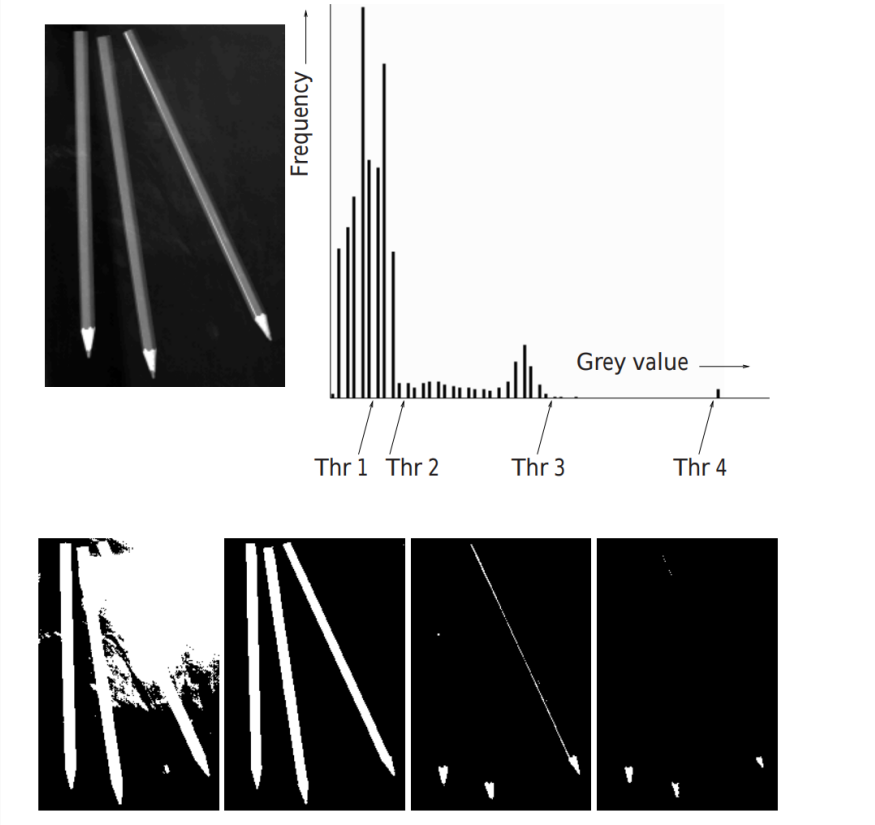


Сегментация: классические методы

- Сегментацию уже 50 лет развивают. Классические методы:
  - thresholding: выбираем порог и превращаем картинку в бинарную;
  - вариант – region growing: выбираем начальные пиксели и постепенно растим регионы вокруг них, останавливаясь на каком-то пороге;
  - кластеризация: кластеризуем пиксели; определение расстояния обычно основано как на свойствах пикселя, так и на его местоположении;
  - сейчас поговорим подробнее...

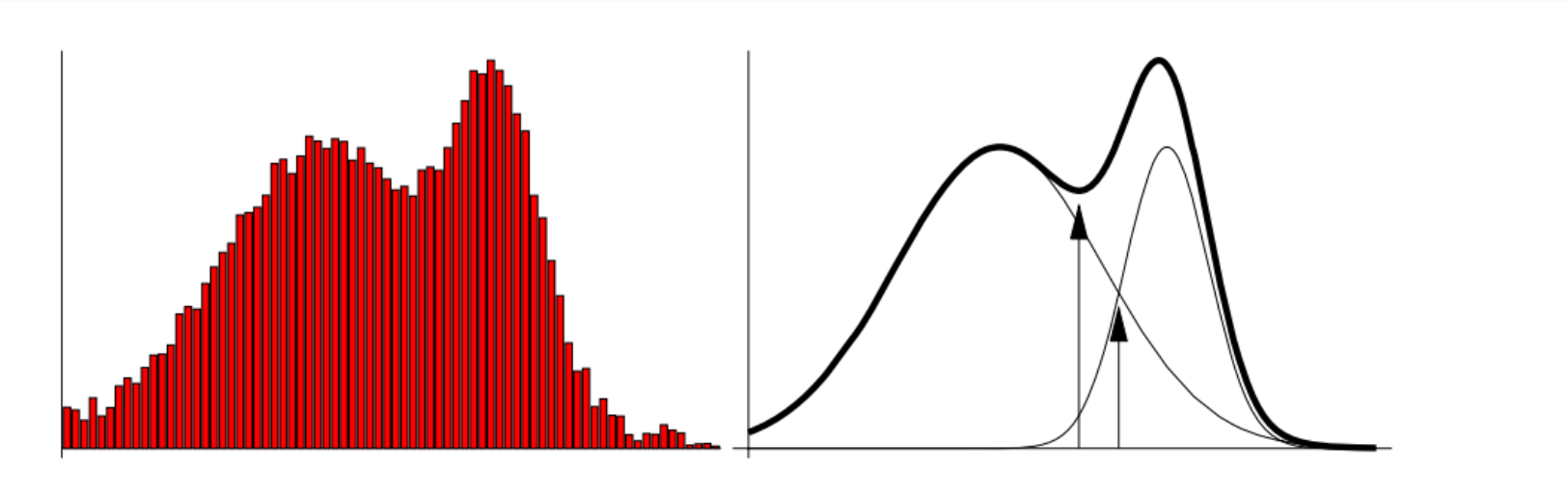
Сегментация: классические методы

- Сегментацию уже 50 лет развивают. Классические методы:
  - всё это известно как histogram-based segmentation: выбираем пороги на основе гистограммы пикселей; даже один порог можно выбирать по-разному:



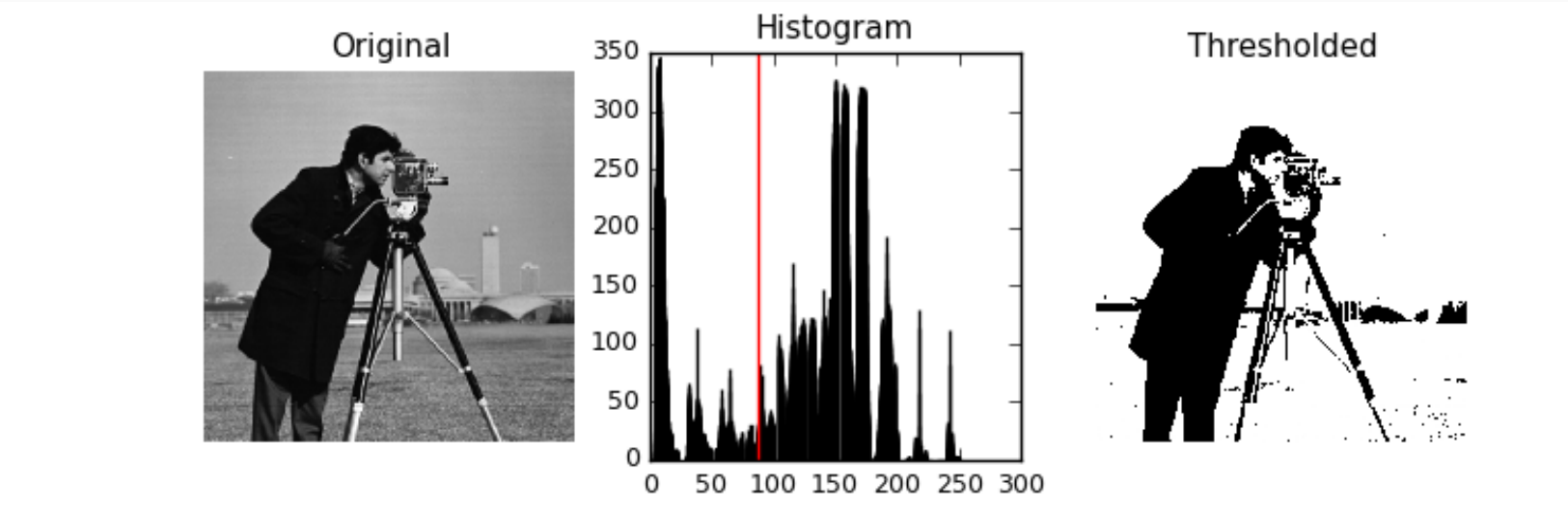
Сегментация: классические методы

- Сегментацию уже 50 лет развивают. Классические методы:
  - можно сделать предположения о пиках в гистограмме и пытаться находить оптимальные пороги:



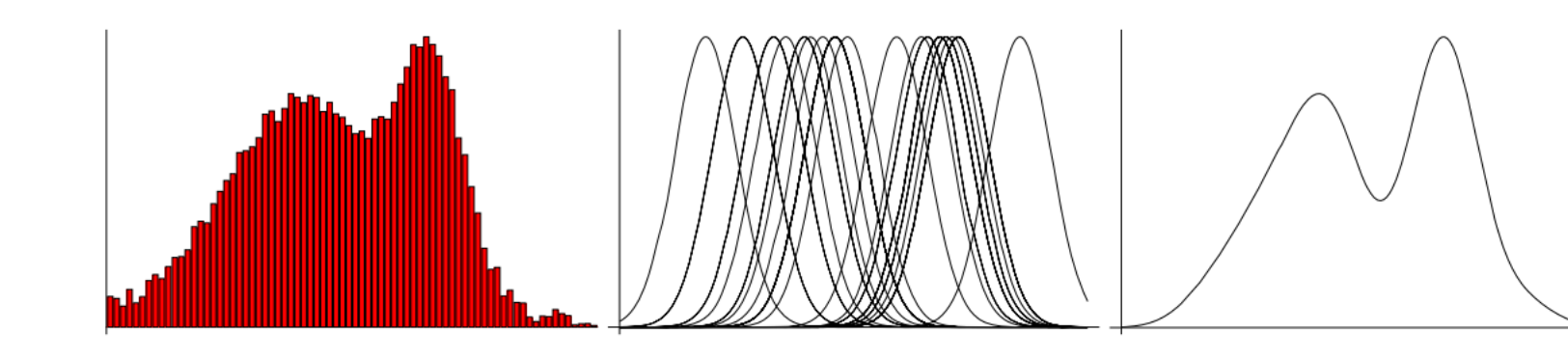
Сегментация: классические методы

- Сегментацию уже 50 лет развивают. Классические методы:
  - метод Оцу (Otsu's method) — выбираем порог, минимизируя суммарную внутриклассовую дисперсию:



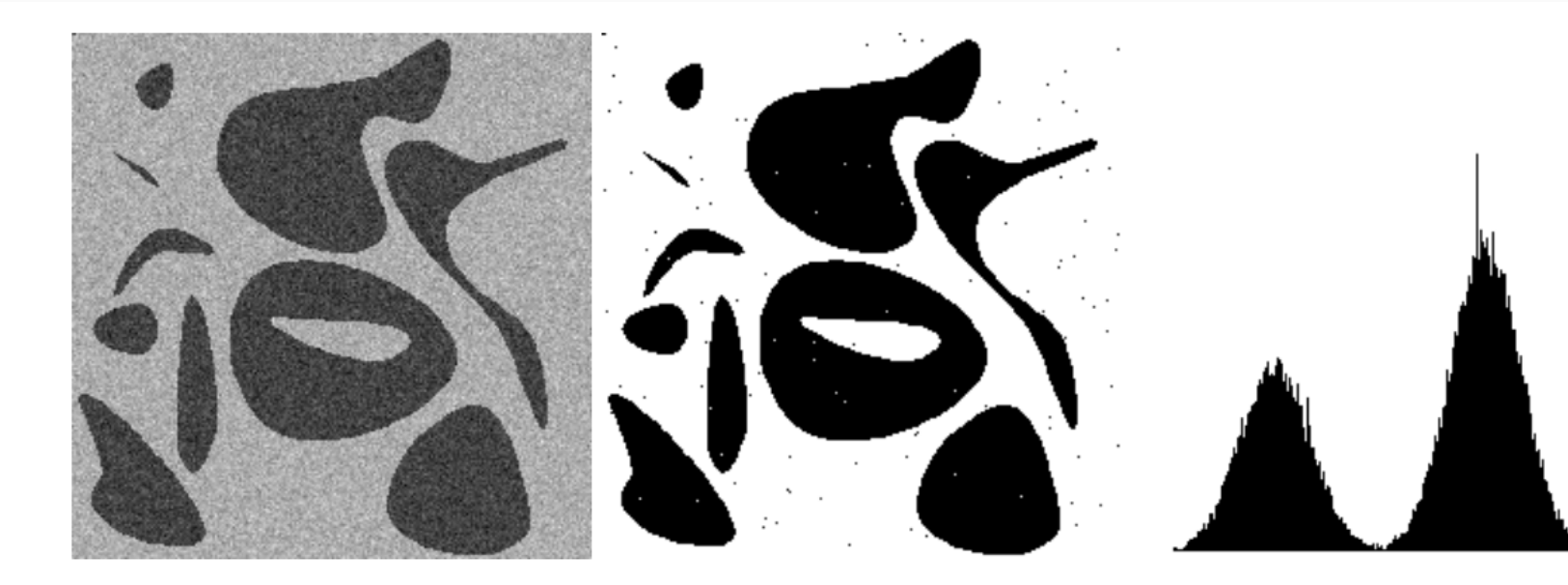
Сегментация: классические методы

- Сегментацию уже 50 лет развивают. Классические методы:
  - можно сделать непрерывную гистограмму методом парзеновского окна (Parzen window) – посчитаем $p(x) = \frac{1}{N} \sum_i d(x - x_i)$:



Сегментация: классические методы

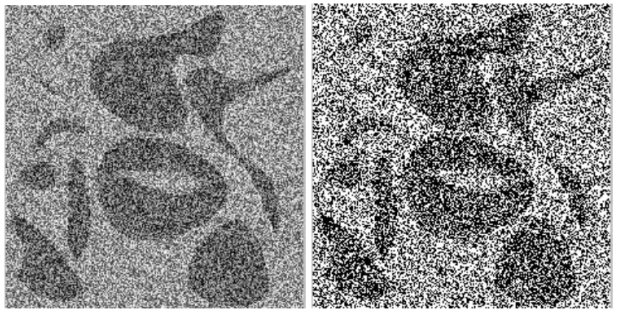
- Сегментацию уже 50 лет развивают. Классические методы:
  - но всё это страдает от того, что шум делает ошибки в обычном thresholding неизбежными:



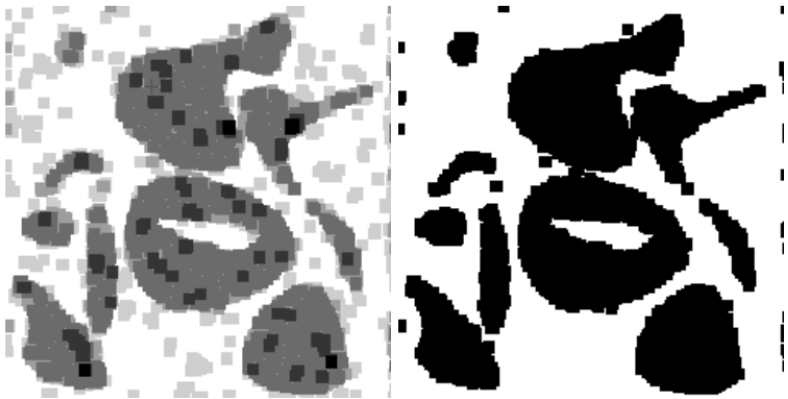
  - тут можно сделать closing на результатах, или ещё проще, медианный фильтр на результате thresholding

Сегментация: классические методы

- Сегментацию уже 50 лет развивают. Классические методы:
  - например, с большим шумом thresholding не работает совсем:

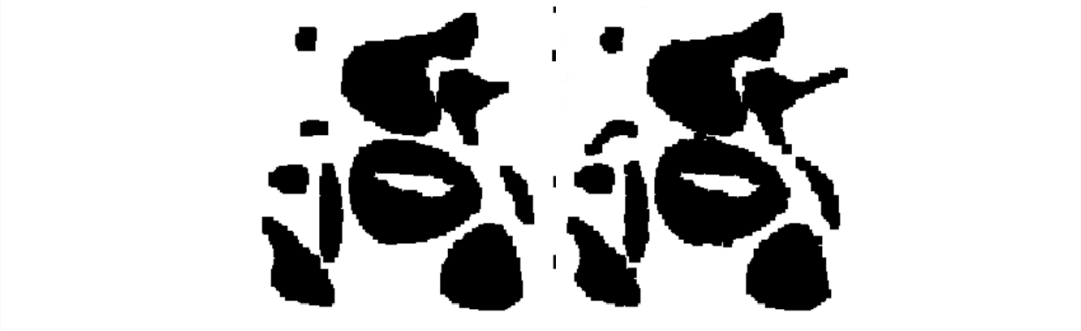


  - после медианного фильтра какие-то квадратики; справа удалили сегменты со слишком маленькими значениями:

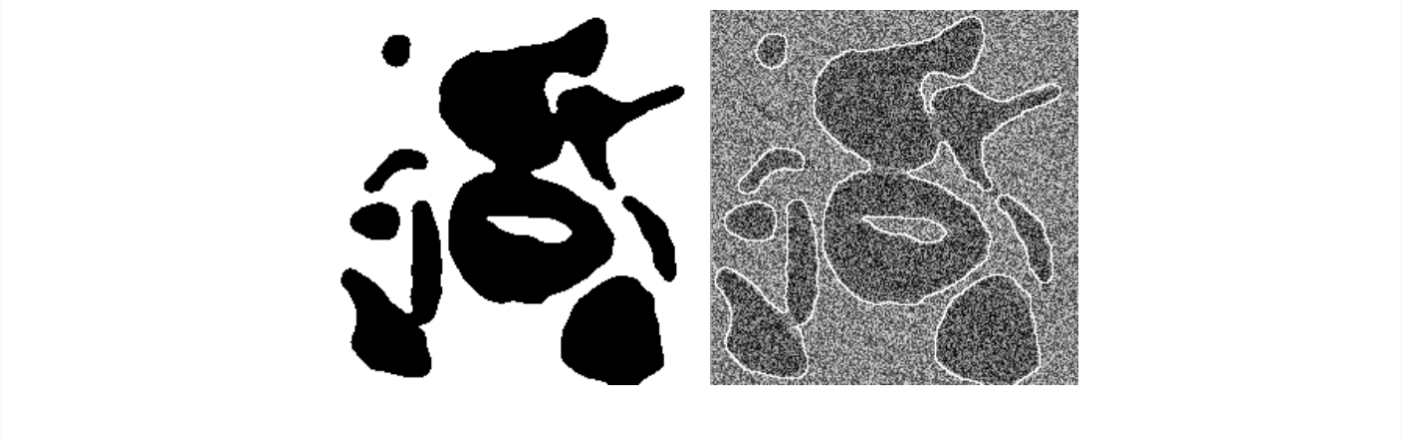


Сегментация: классические методы

- Сегментацию уже 50 лет развивают. Классические методы:
  - затем можно сделать closing (слева) и удалить слишком маленькие сегменты по площади:



  - потом добавим диффузию, и результат уже хорошо смотрится:

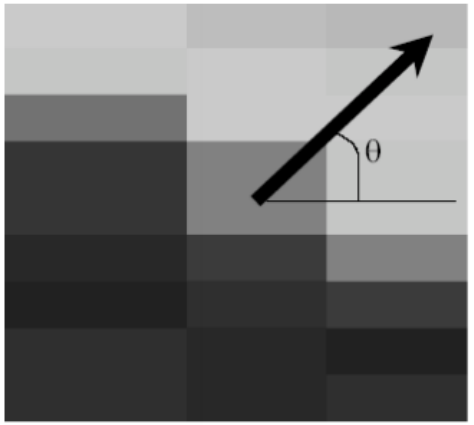


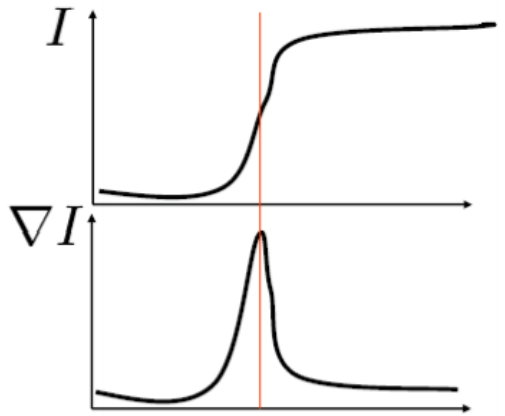
Сегментация: классические методы

- Сегментацию уже 50 лет развивают. Классические методы:
  - edge-based segmentation; вместо порогов на пиксели будем искать границы между сегментами:
    - найдём все правдоподобные границы на картинке;
    - оставим только замкнутые границы потенциальных сегментов;
    - заполним их, получив сегментацию;
  - самая сложная здесь в присутствии шума – вторая часть;

Сегментация: классические методы

- Сегментацию уже 50 лет развивают. Классические методы:
  - edge detection делается градиентными методами:





Edge pixels are at local maxima of gradient magnitude
Gradient direction is always perpendicular to edge direction

Gradient Vector: $\nabla I = [\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}]^T$

Magnitude: $|\nabla I| = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2}$

Orientation $\theta = atan2(\frac{\partial I}{\partial y}, \frac{\partial I}{\partial x})$

Сегментация: классические методы

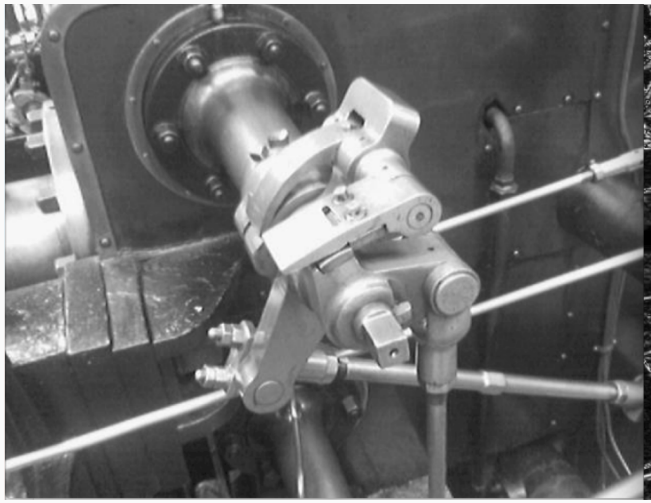
- Сегментацию уже 50 лет развивают. Классические методы:
  - главный метод для вычисления градиента – оператор Собеля, свёртка 3×3:

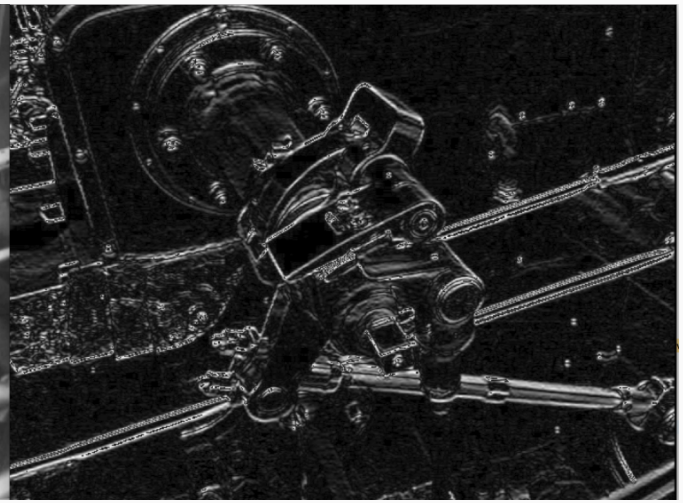
-1	0	+1
-2	0	+2
-1	0	+1

x filter

+1	+2	+1
0	0	0
-1	-2	-1

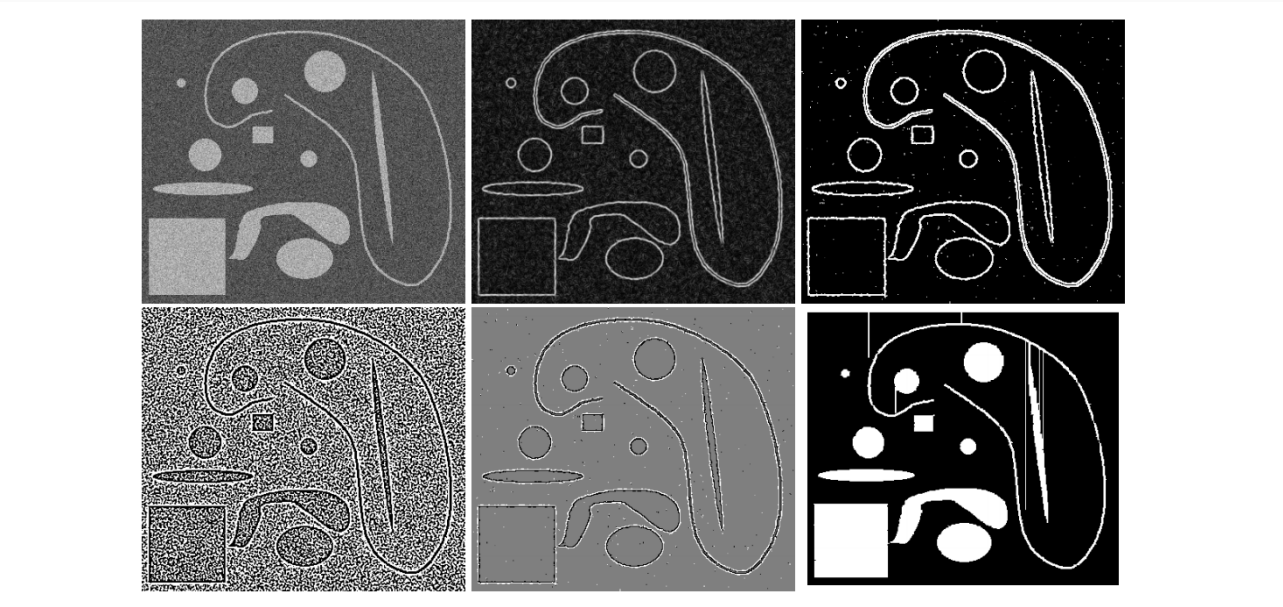
y filter





Сегментация: классические методы

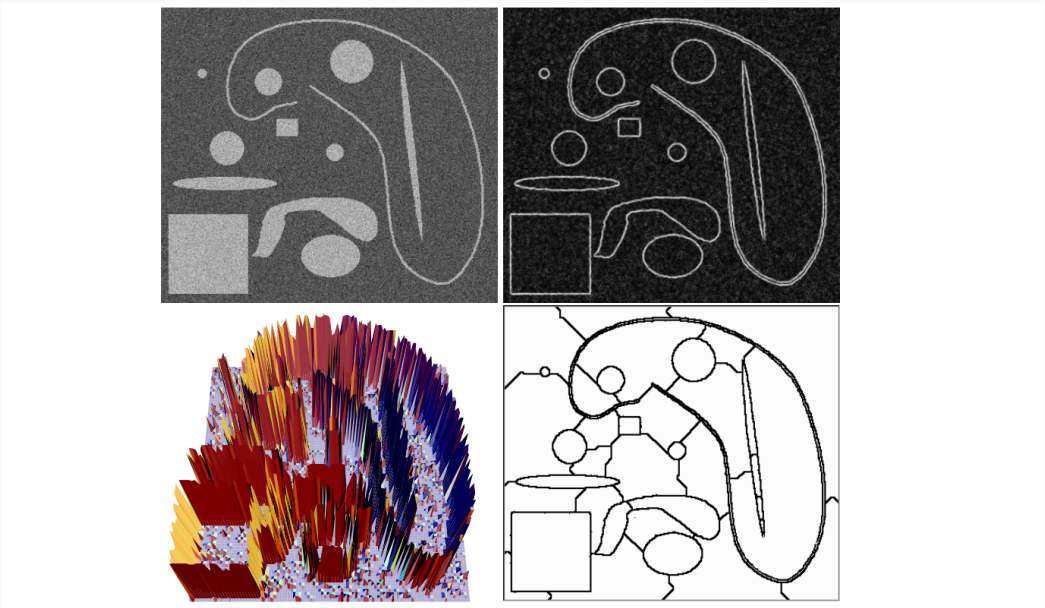
- Сегментацию уже 50 лет развивают. Классические методы:
  - и если картинка была хорошая, то всё хорошо; а с шумом не очень:



Слева направо, сверху вниз: исходная картинка, edgeness image (модуль градиента), его thresholding, знак лапласиана (показывает, куда меняется градиент), произведение thresholded edge image и лапласиана, заполнение контуров.

Сегментация: классические методы

- Сегментацию уже 50 лет развивают. Классические методы:
  - watershed segmentation: заполняем картинку ”водой”, начиная от локальных минимумов, строим дамбы там, где волны сталкиваются



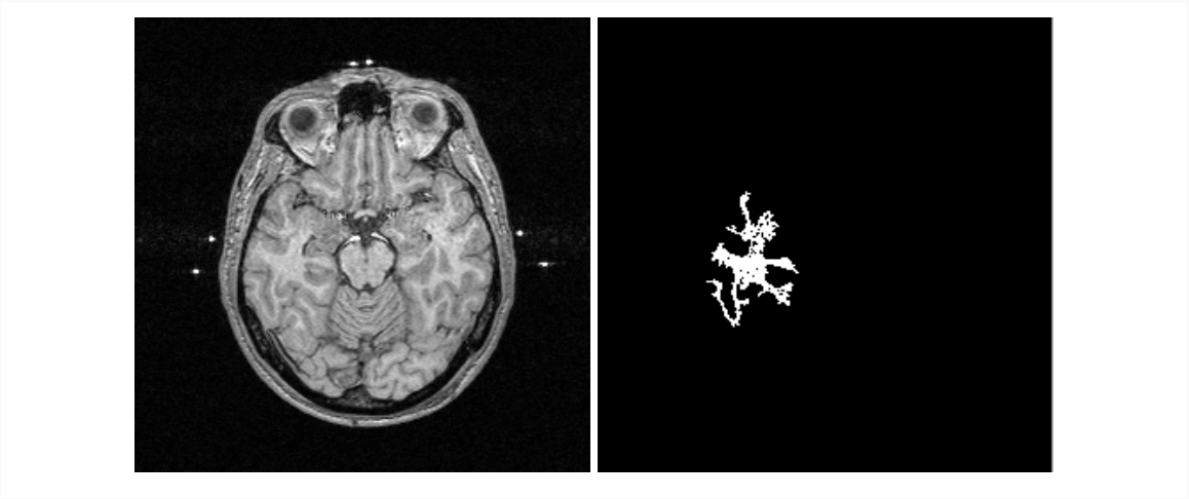
  - но потом надо делать postprocessing, потому что наверняка будут лишние рёбра;

Сегментация: классические методы

- Сегментацию уже 50 лет развивают. Классические методы:
  - region-based segmentation:
    - сначала сделаем очень подробную сегментацию на маленькие кусочки;
    - затем будем постепенно кусочки объединять;
  - начальная сегментация теоретически вообще может быть по одному пикселю;

Сегментация: классические методы

- Сегментацию уже 50 лет развивают. Классические методы:
  - region growing:



  - масса разных метрик для оценки того, какие сегменты объединять; или, наоборот, обратно разъединять (split & merge methods)...
  - ещё масса методов для определения текстур, но это уже точно не будем.

Сегментация: классические методы

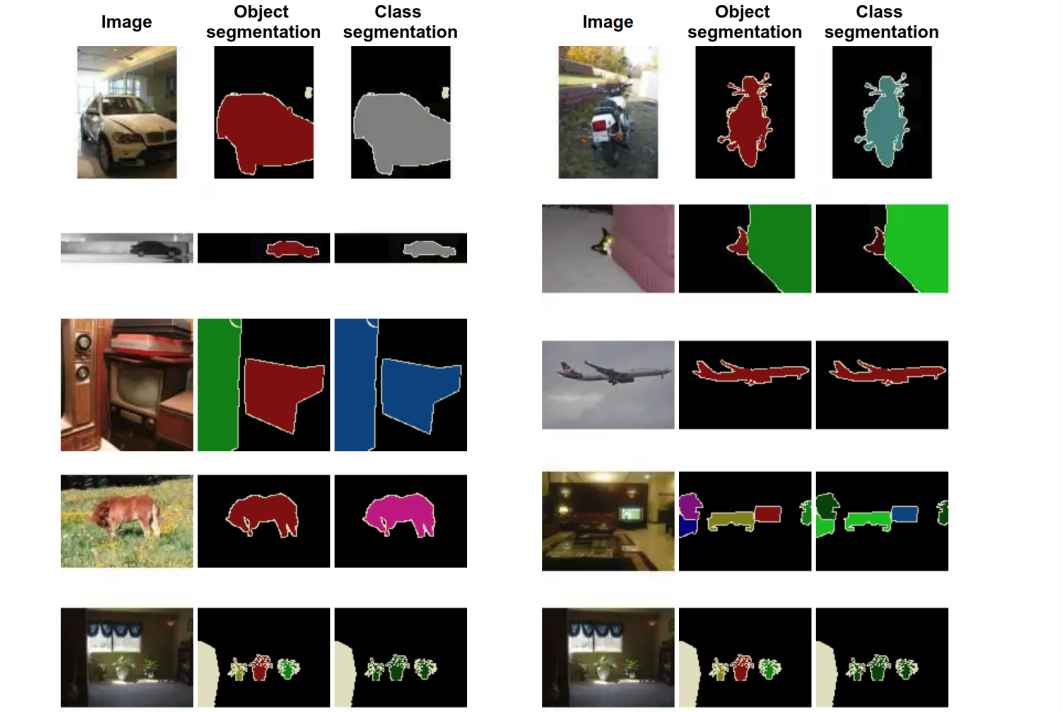
- В прошлый раз, кстати, видели классический алгоритм selective search
- (Felzenszwalb, Huttenlocher, 2004): Efficient Graph-Based Image Segmentation
  - представляем картинку в виде графа, в котором вес на ребре между вершинами показывает их похожесть;
  - придумываем предикат для сравнения разных компонентов графа;
  - так получаются, при определённых параметрах, базовые маленькие кусочки.





Датасеты

- Кстати, про циферки – откуда они берутся в сегментации?
- Результаты обычно оценивают как mIoU
- Pascal VOC: в 2012 было 20 классов, около 7000 сегментаций



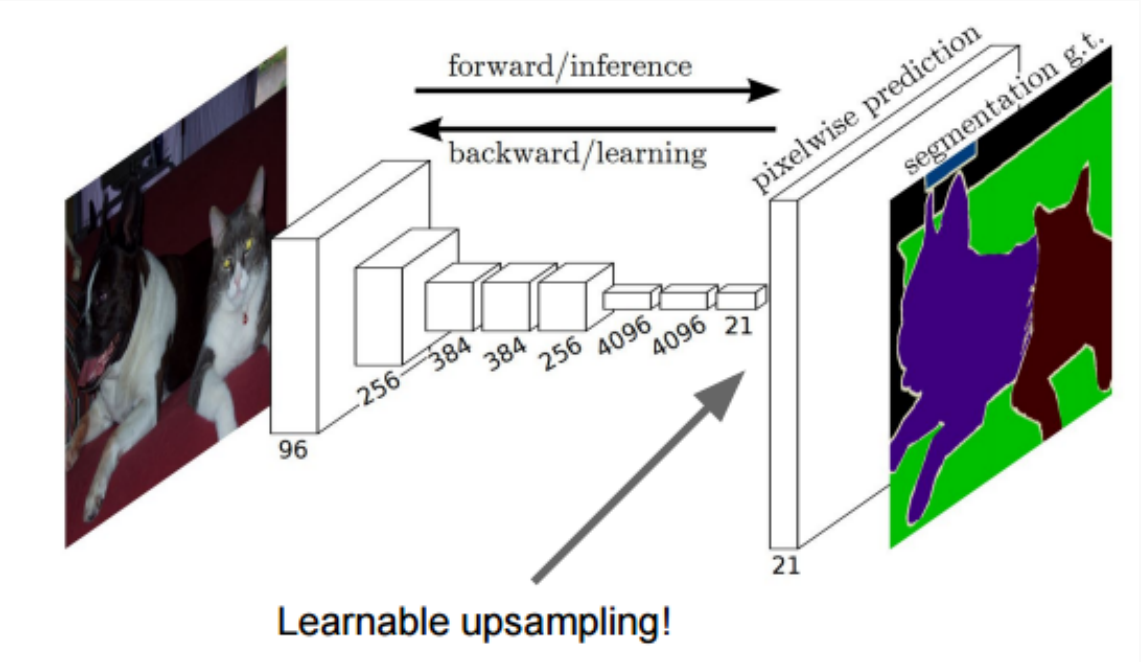
Датасеты

- Более сложный и современный датасет – COCO (а сейчас OpenImages):



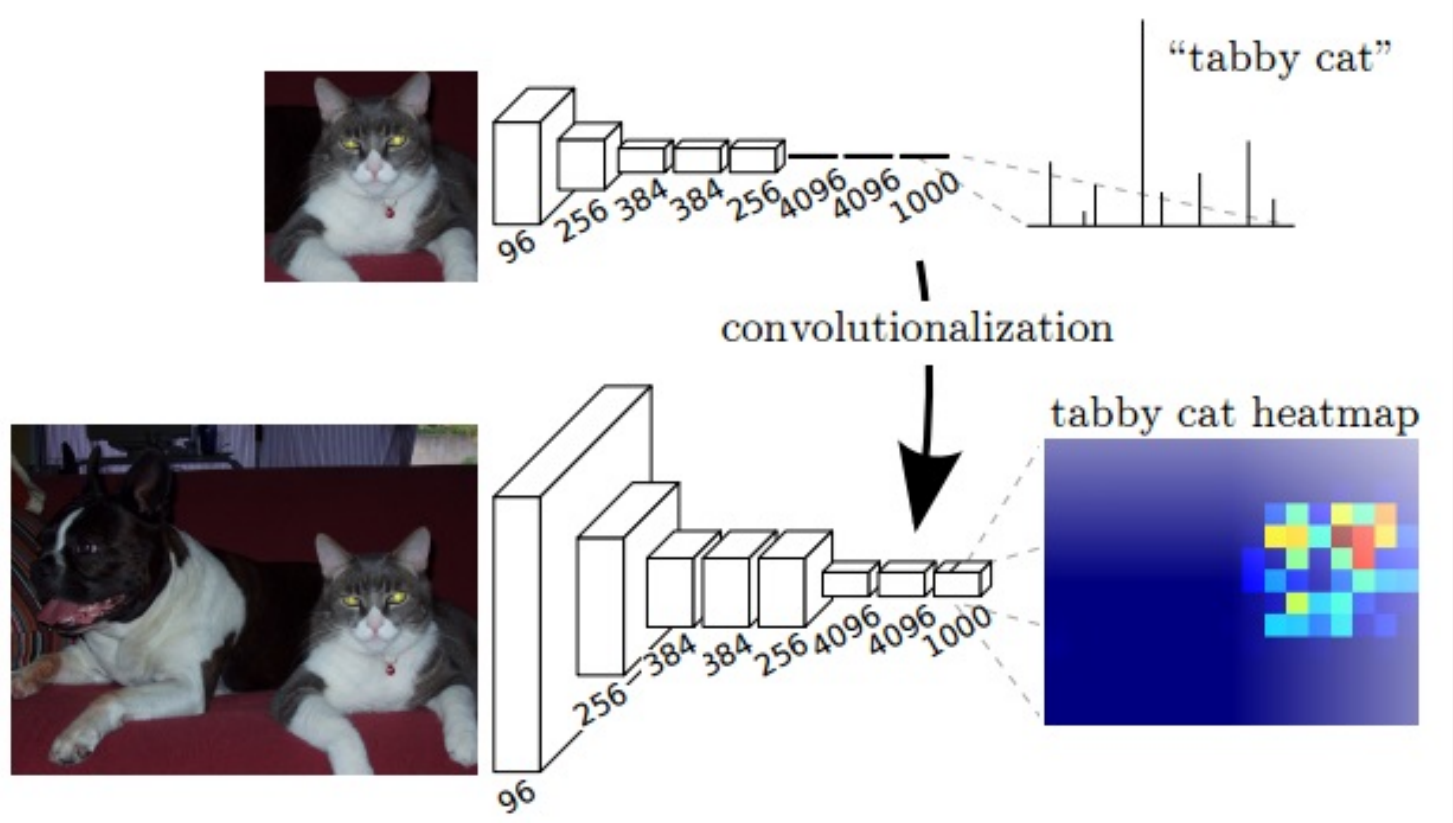
Fully Convolutional Networks

- (Long et al., 2015): Fully Convolutional Networks for Semantic Segmentation
- FCN (fully convolutional network): вместо полносвязных уровней добавляем свёртки с большим receptive field (upsampling).



Fully Convolutional Networks

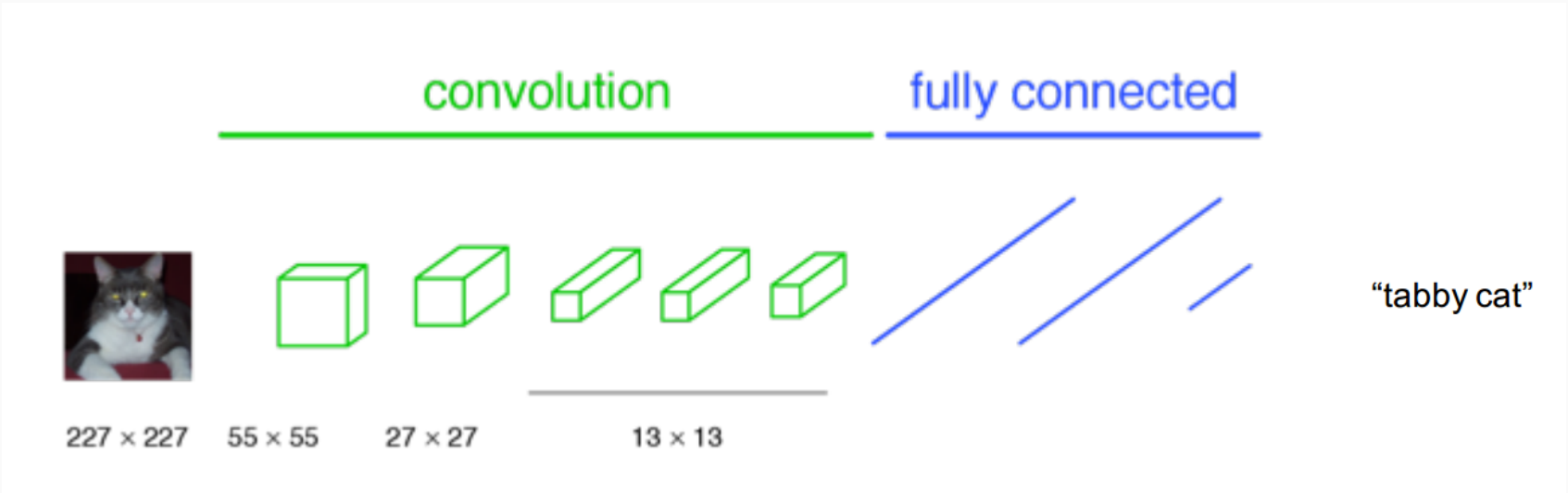
- Эффект локализации получается за счёт того, чтобы посмотреть, где более сильные активации:



- Идея в том, чтобы этот эффект отмасштабировать обратно к исходному размеру картинки.

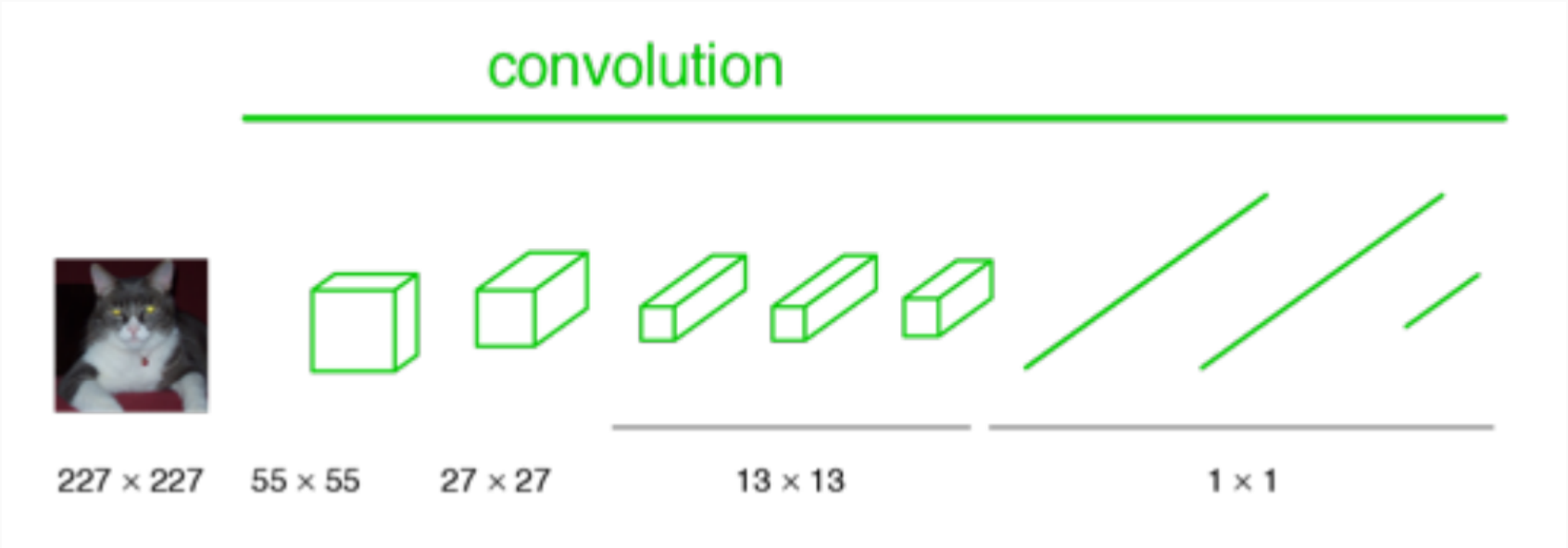
Fully Convolutional Networks

- И можно сконвертировать “обычную” CNN в FCN – начнём с обычной CNN для классификации:



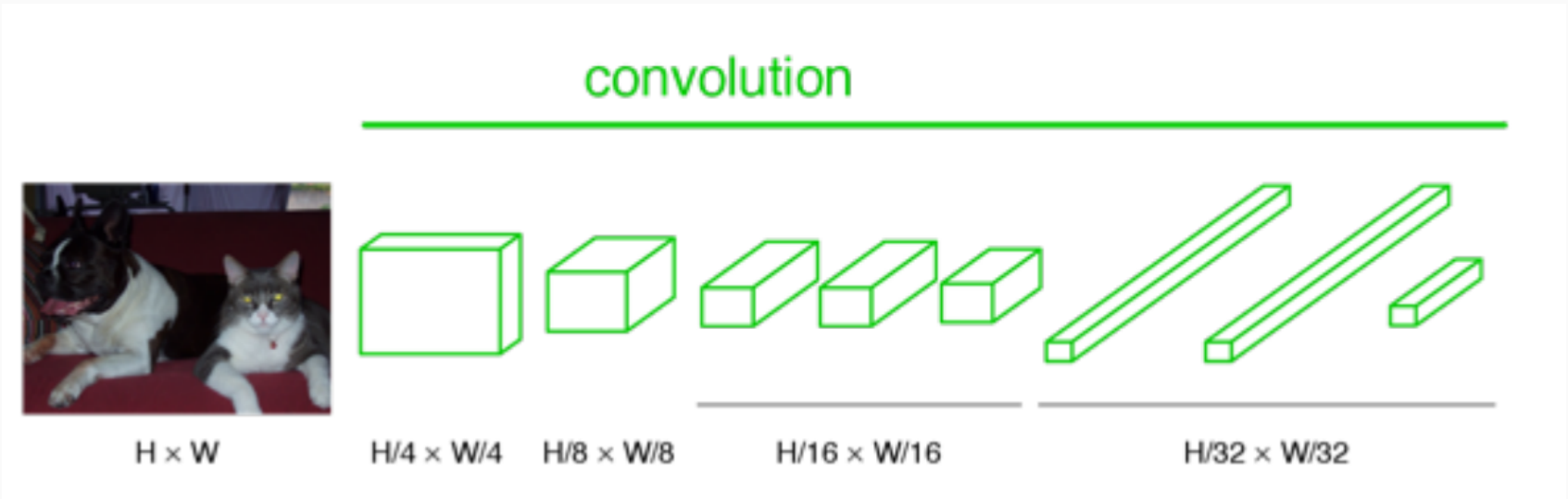
Fully Convolutional Networks

- Конвертируем полносвязные слои в 1×1 свёрточные слои (с теми же весами):



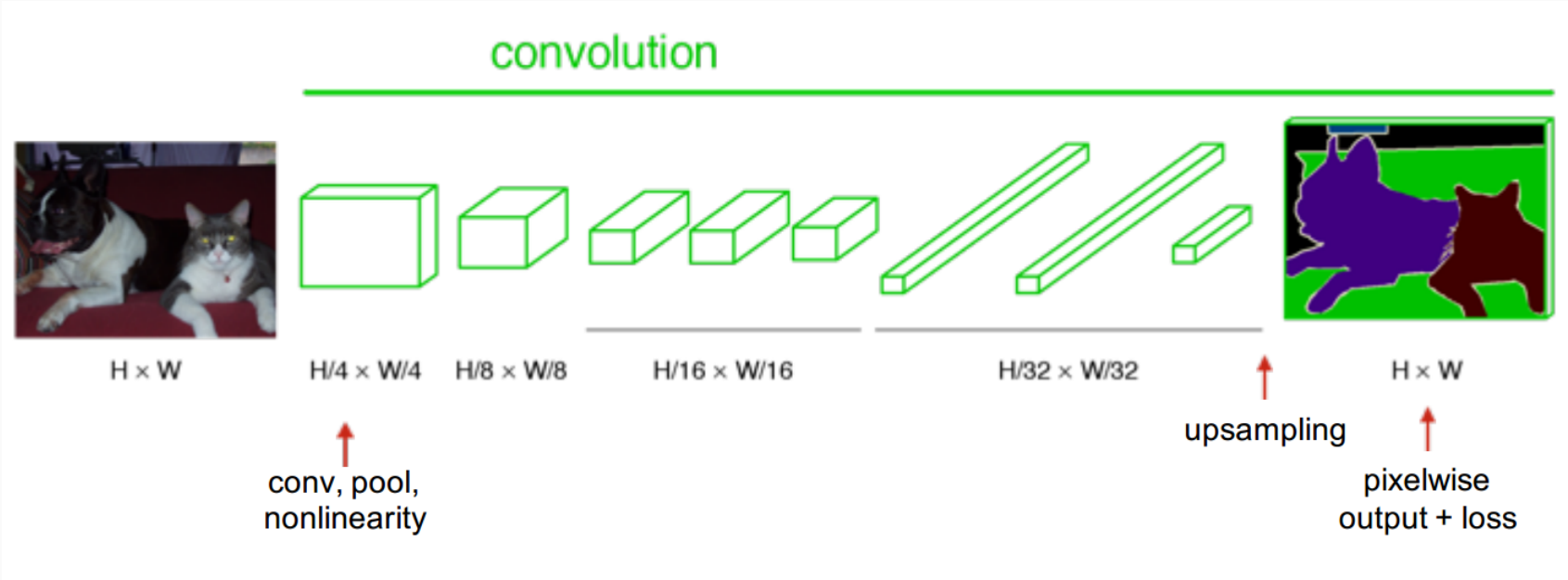
Fully Convolutional Networks

- Теперь вход может быть, кстати говоря, переменного размера:



Fully Convolutional Networks

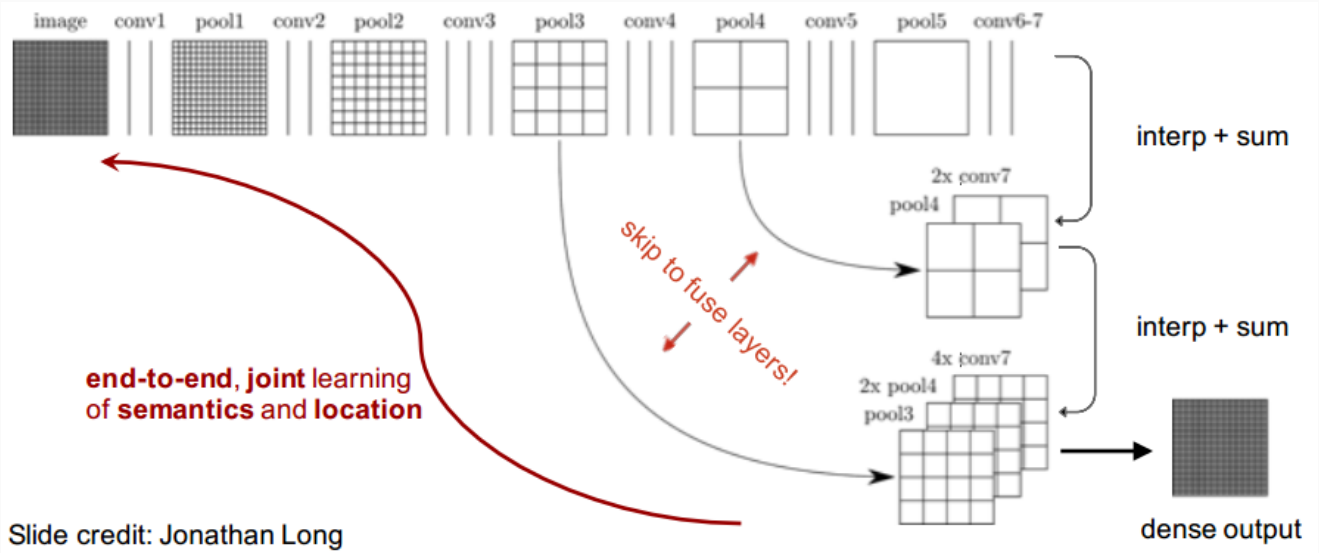
- А для upsampling добавляет деконволюцию (транспонированную свёртку):

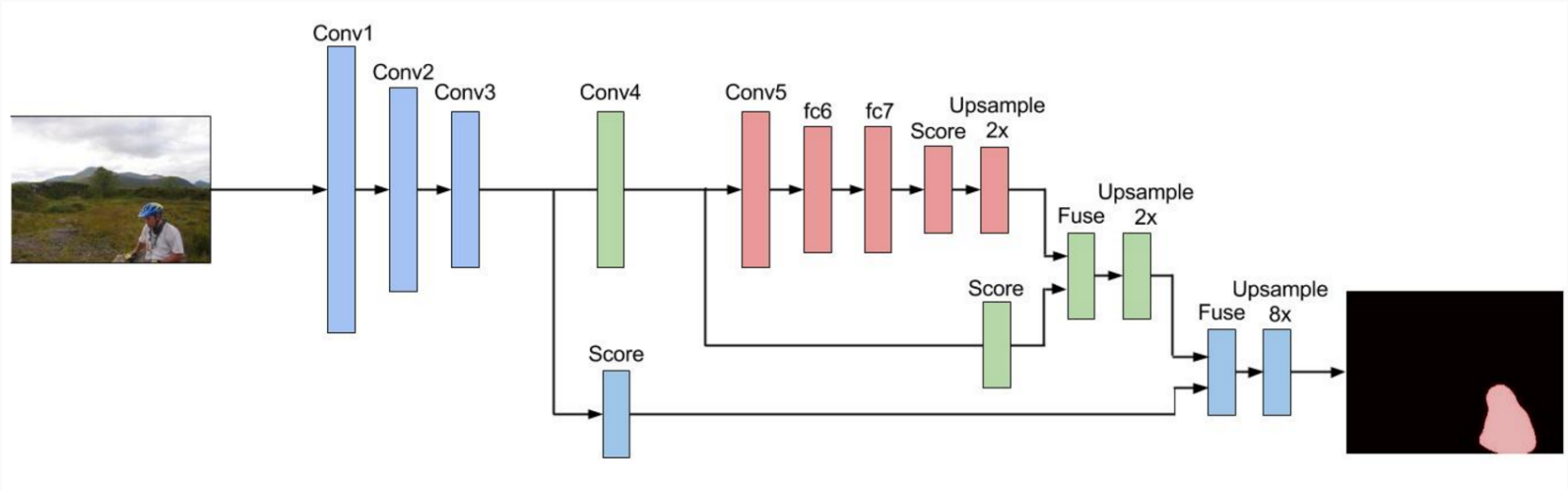


- Проблема здесь в том, что мы, конечно, теряем разрешение, потому что с активациями сначала был downsampling.

Fully Convolutional Networks

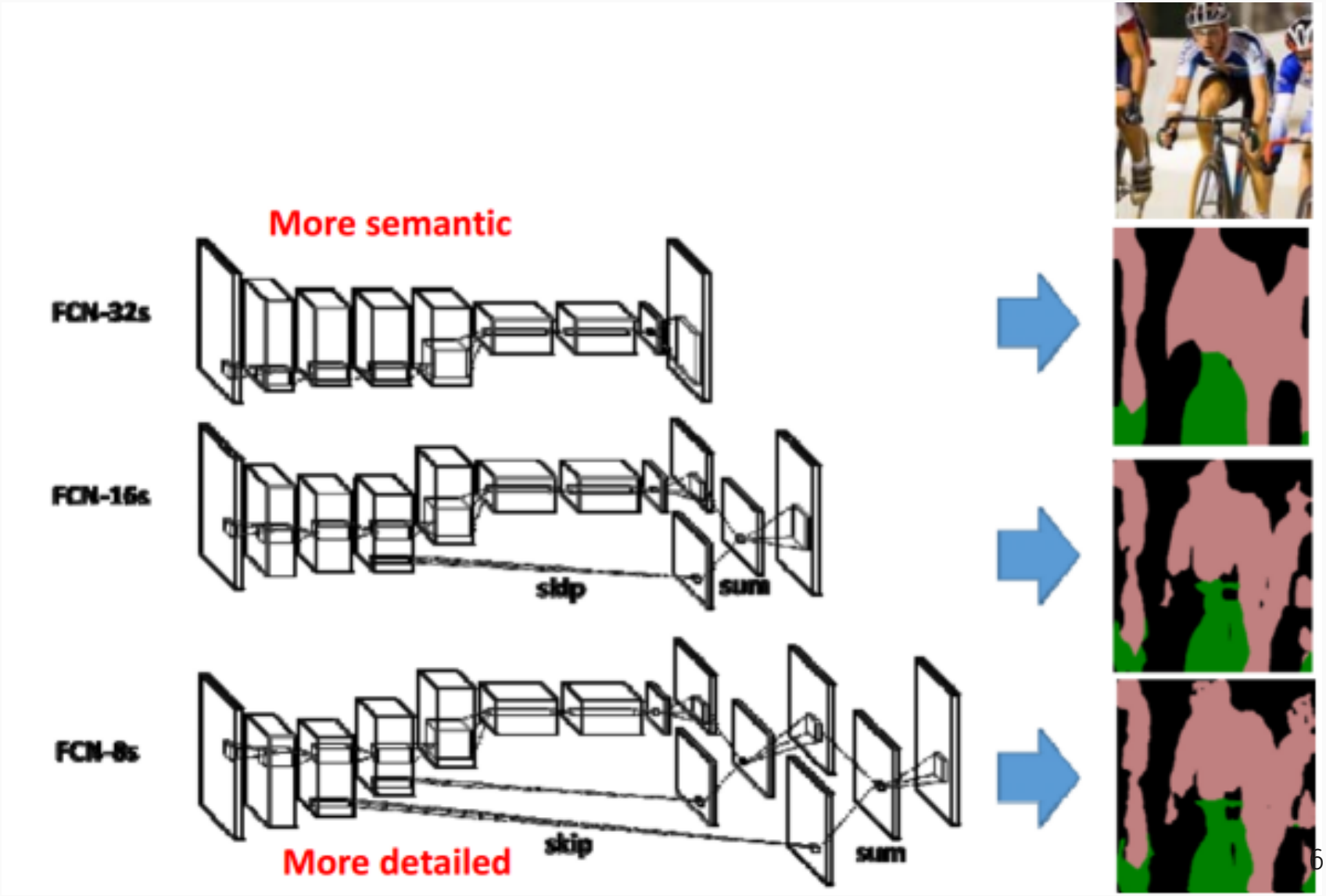
- Решение в том, чтобы добавить skip connections с предыдущих уровней:





Fully Convolutional Networks

- Вот какой эффект дают skip connections:



Fully Convolutional Networks

- Пример результатов FCN:



DeconvNet

- Большое достоинство FCN – берут всю картинку на вход и быстро работают: (Long et al., 2015) был в 286 раз быстрее предшественника (SDS, Simultaneous Detection and Segmentation).
- Проблема FCN – трудно обработать разные масштабы, причём в обе стороны:



(a) Inconsistent labels due to large object size

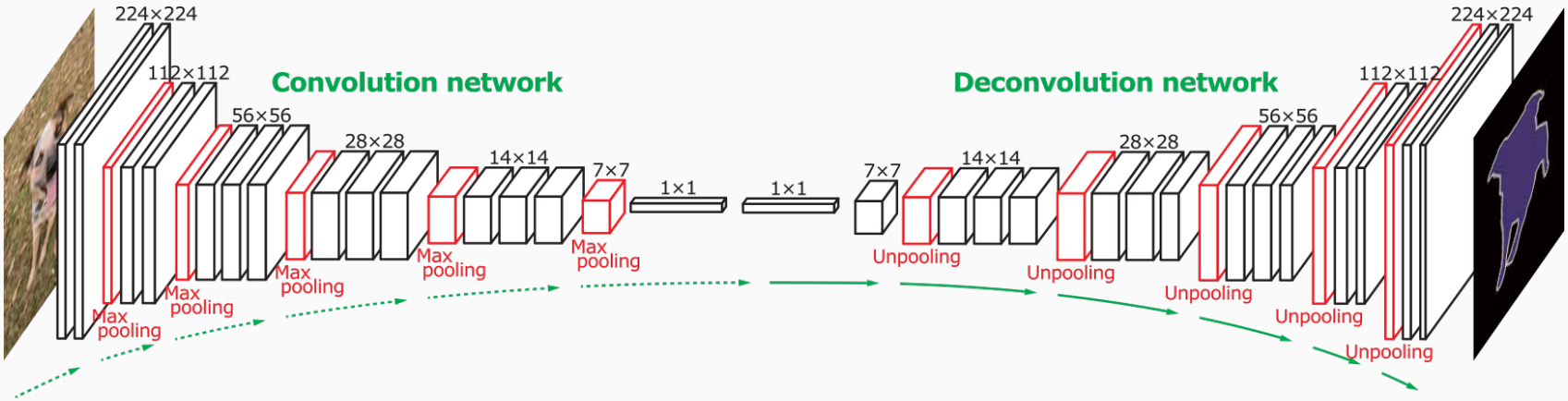


(b) Missing labels due to small object size

- И skip connections хоть и помогают, но не радикально.
- Кроме того, в итоге всё-таки достаточно грубая сегментация

DeconvNet

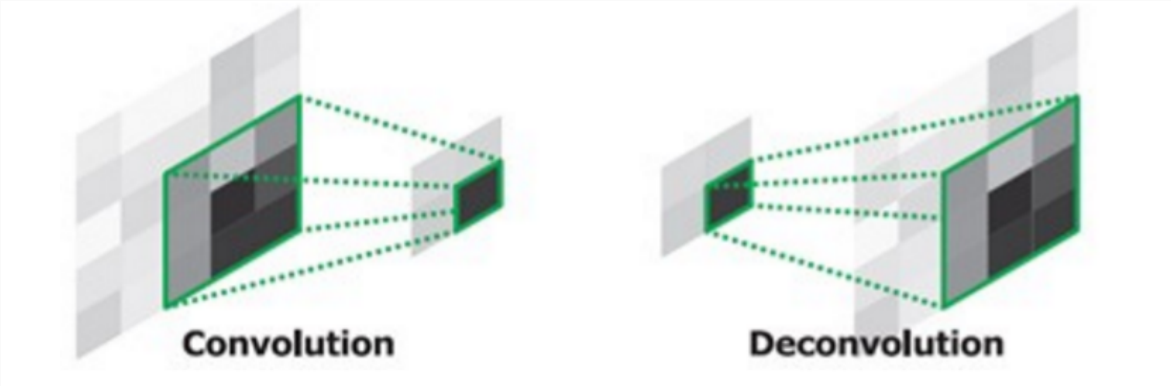
- DeconvNet (Noh et al., 2015) – давайте не делать skip connections, а просто deconvolutions/unpooling всю дорогу, применяя эту сеть к каждому object proposal:

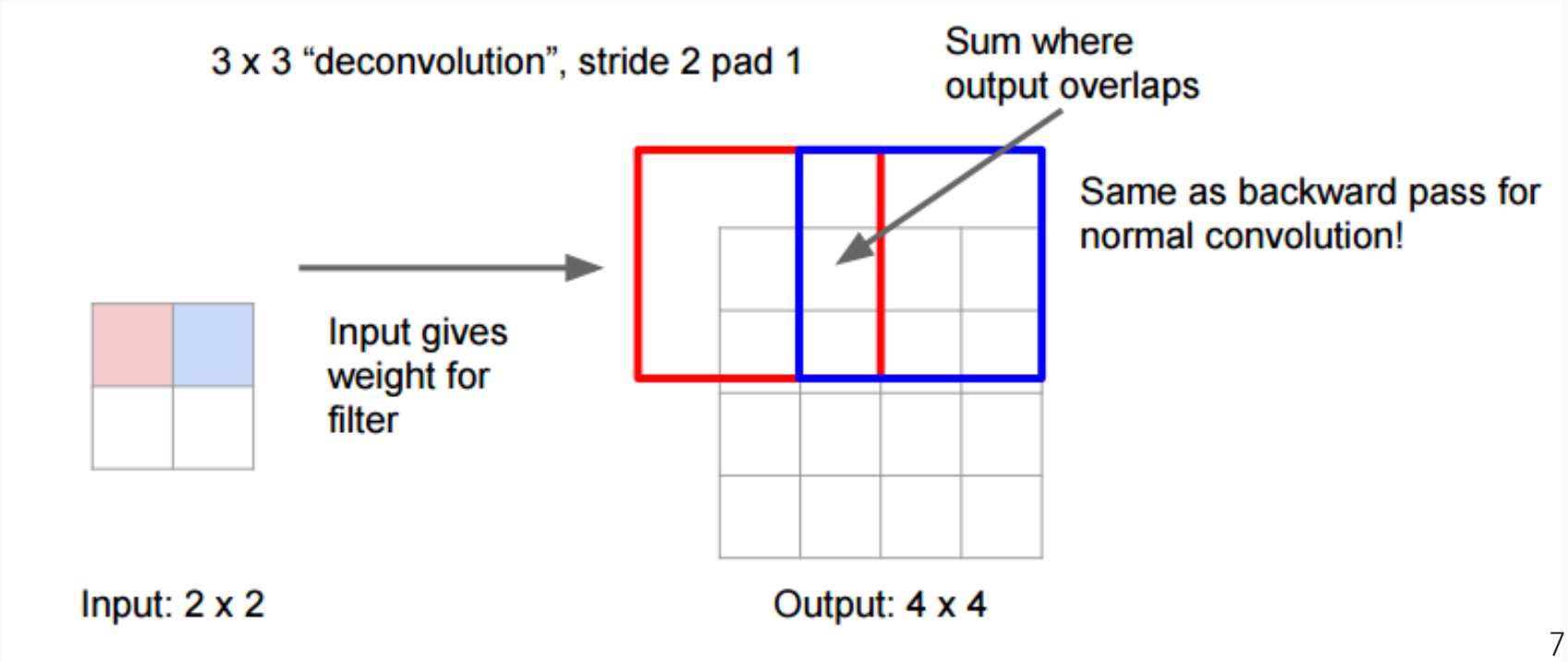


- Деконволюция, unpooling... что это такое вообще?

DeconvNet

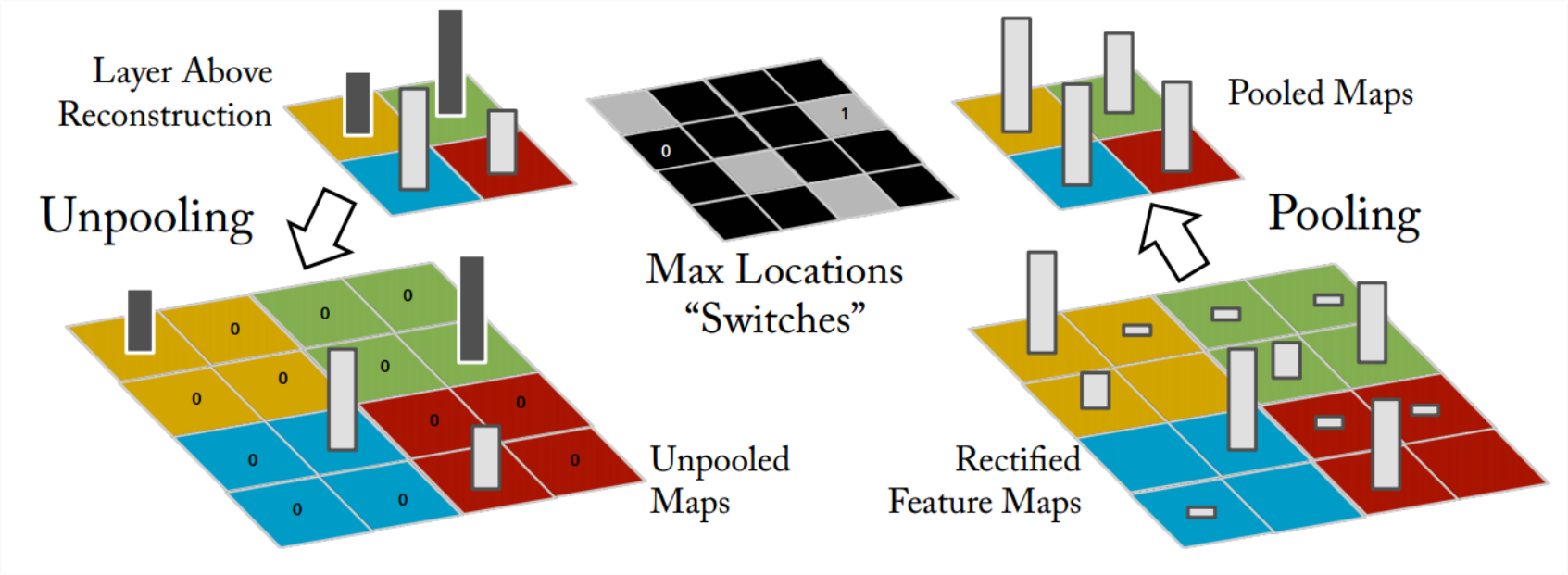
- Деконволюция:

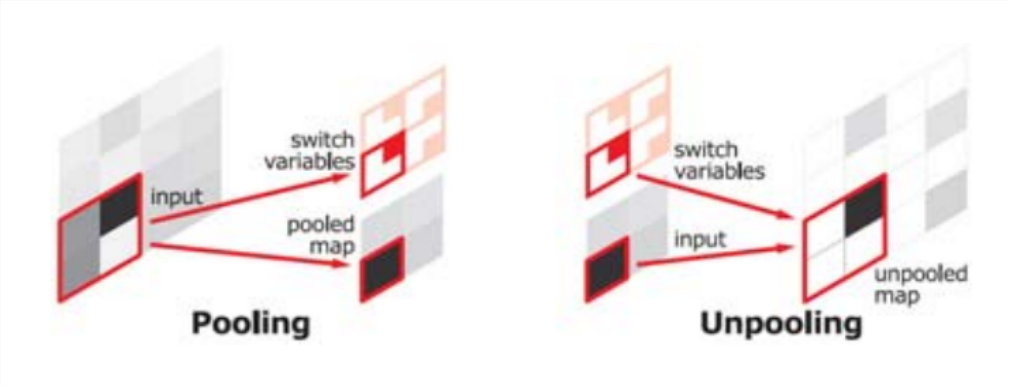




DeconvNet

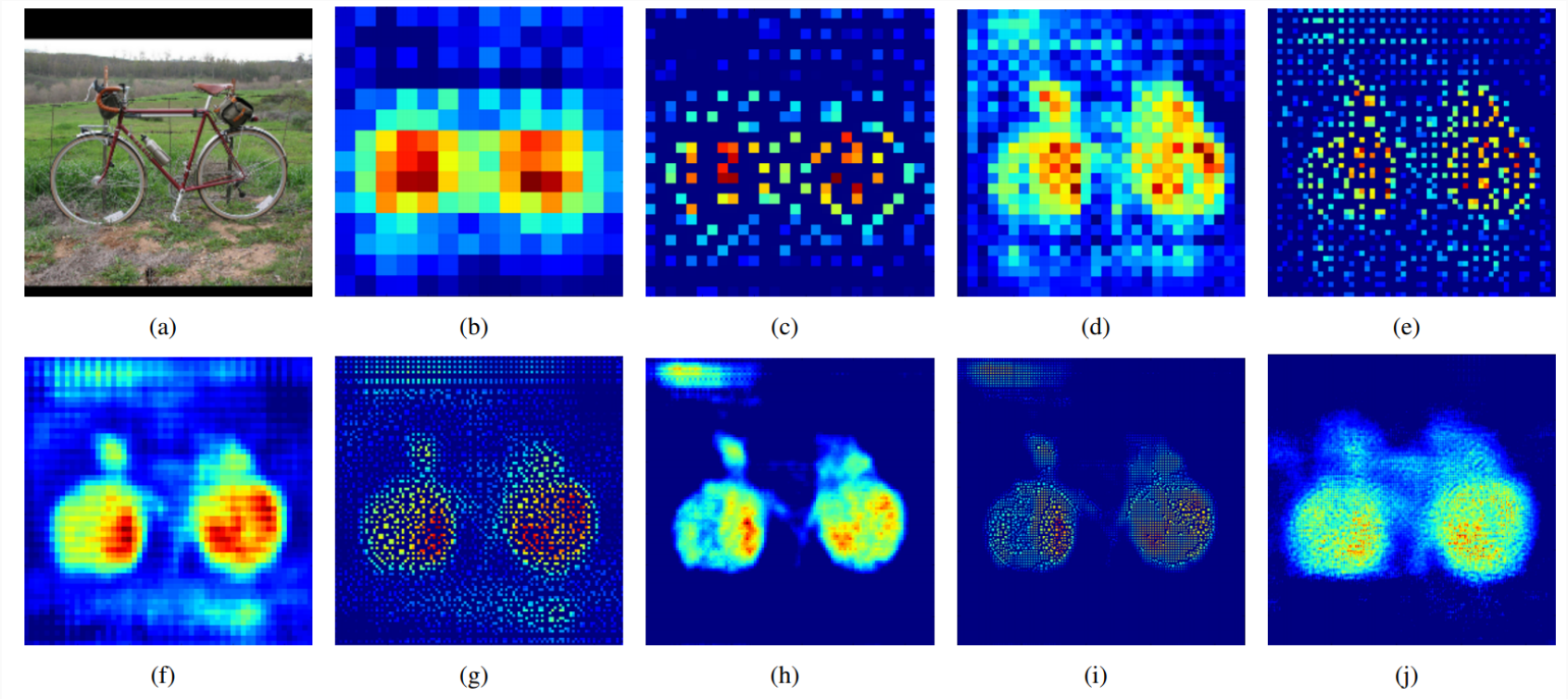
- Unpooling:





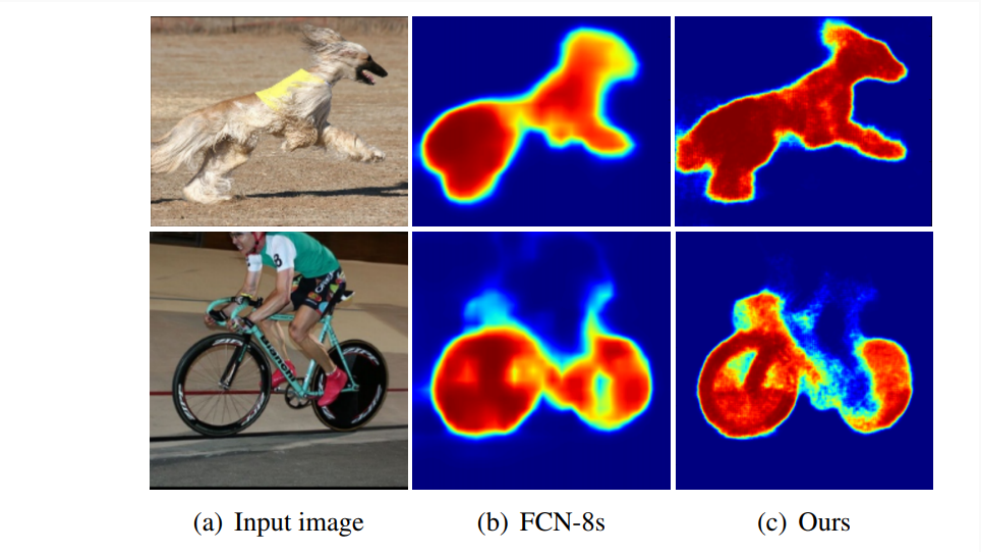
DeconvNet

- Вот так постепенно происходит деконволюция/unpooling:



DeconvNet

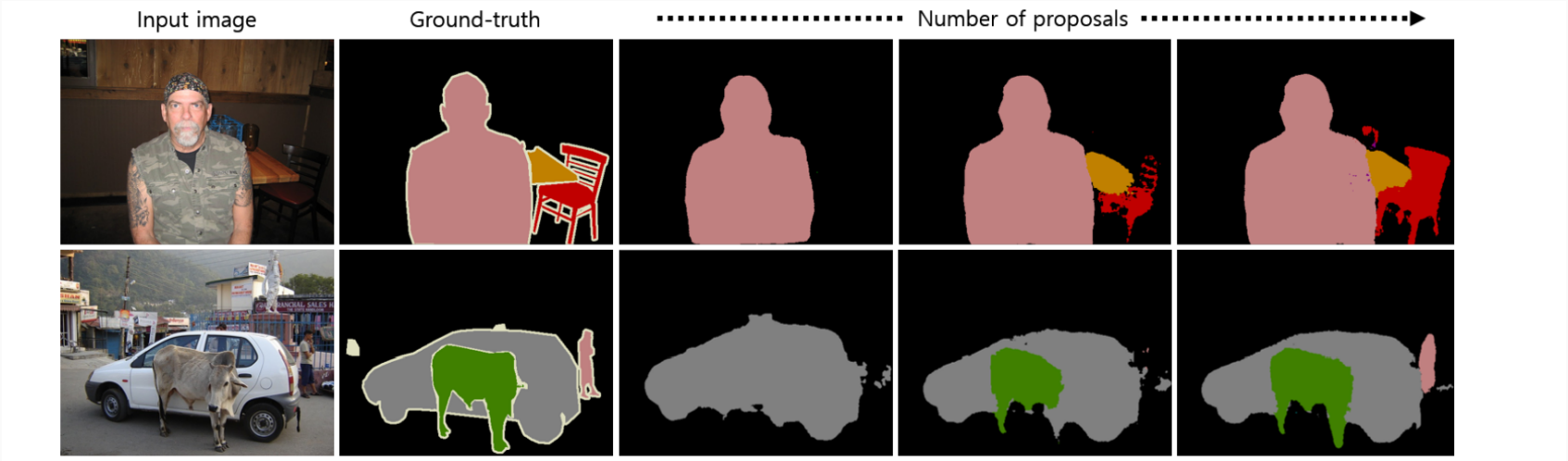
- Дольше обучается, но результаты могут быть лучше, а применение такое же по времени:



(a) Input image (b) FCN-8s (c) Ours

DeconvNet

- Работает отдельно на proposals, их можно упорядочить от большего к меньшему:



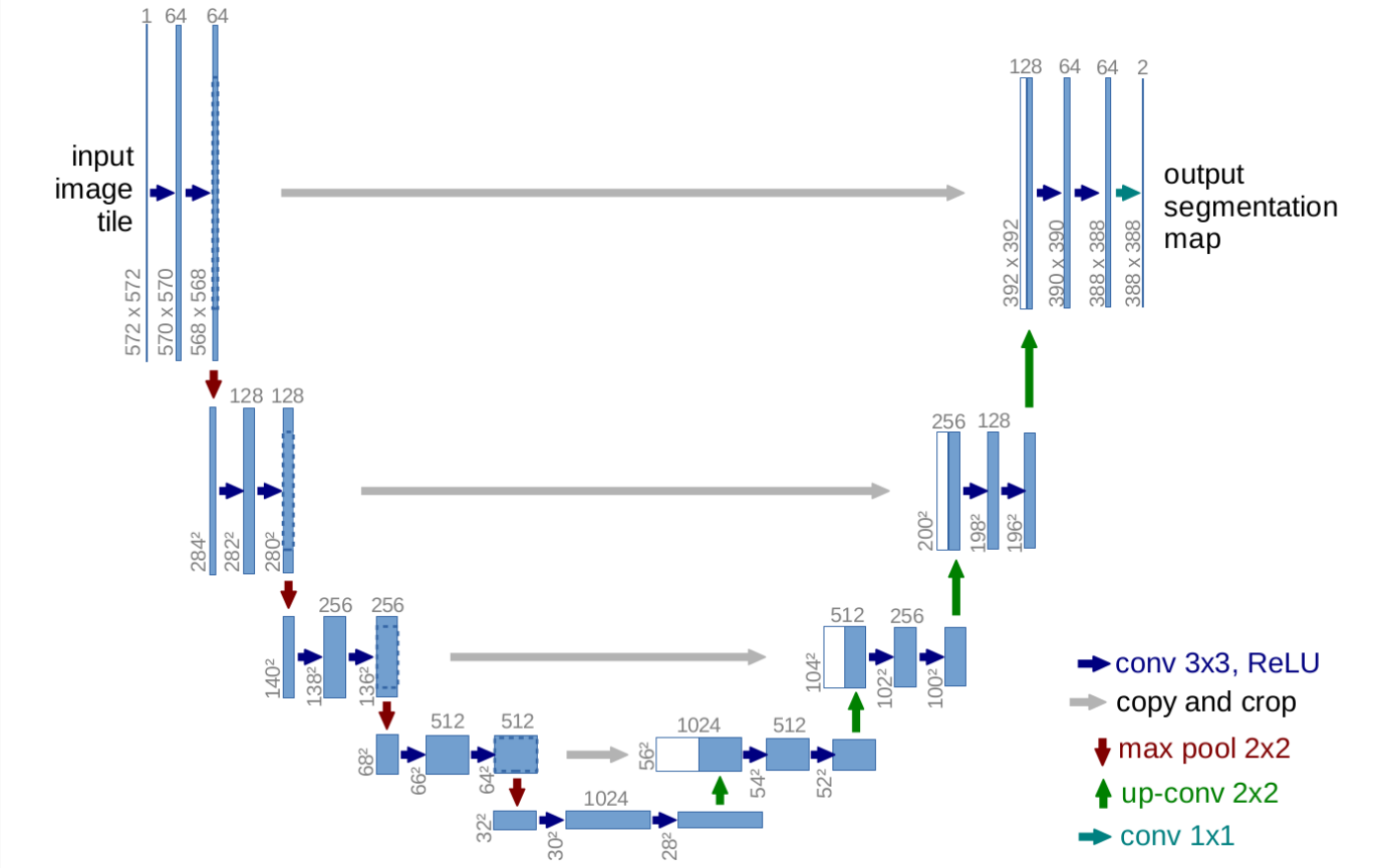
DeconvNet

- А можно сделать ансамбль с FCN:



U-Net

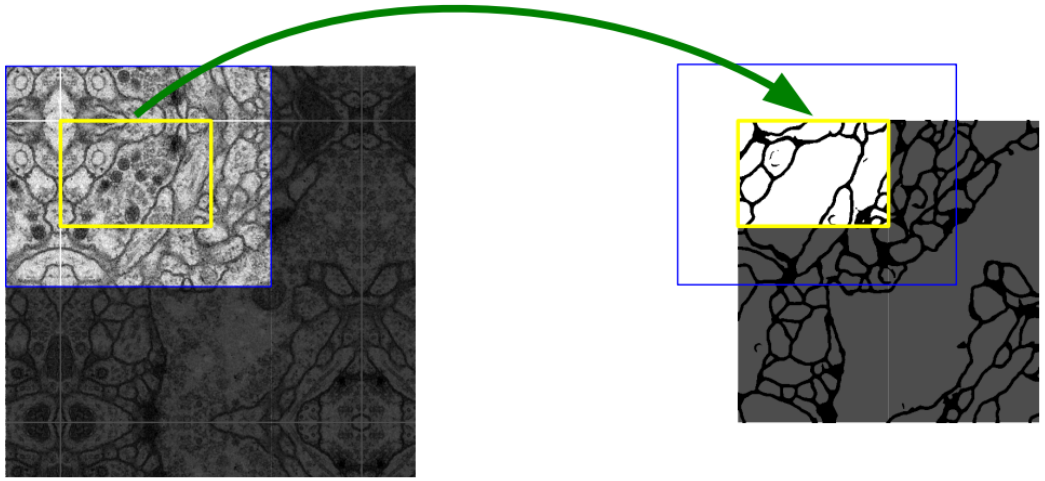
- U-Net (Ronneberger et al., 2015): очень популярная архитектура, особенно в medical imaging.



- Тоже skip connections; теперь upsampling часть по сути симметрична downsampling части, много признаков переходят.

U-Net

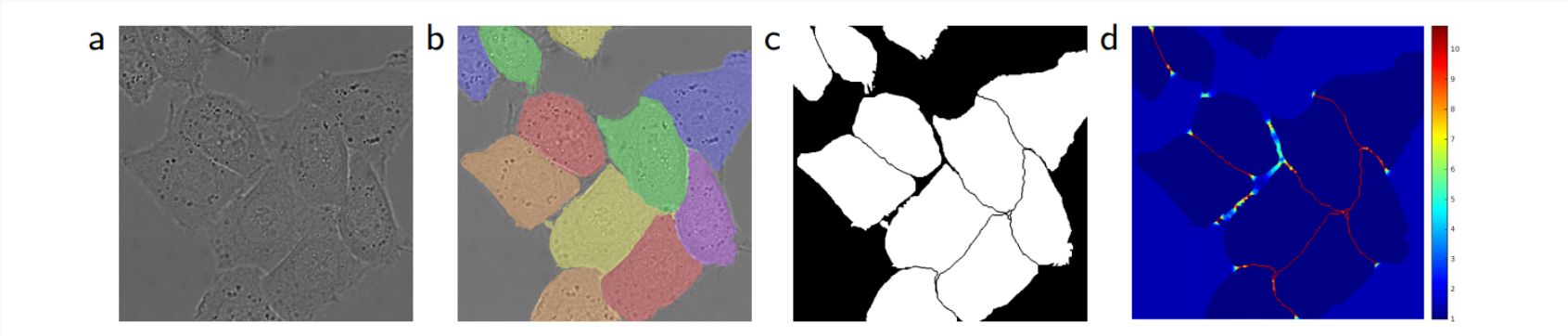
- Может работать с маленькой обучающей выборкой.
- Кроме того, поскольку это fully convolutional сеть, она может работать на картинках любого размера, покрывая их тайлами:



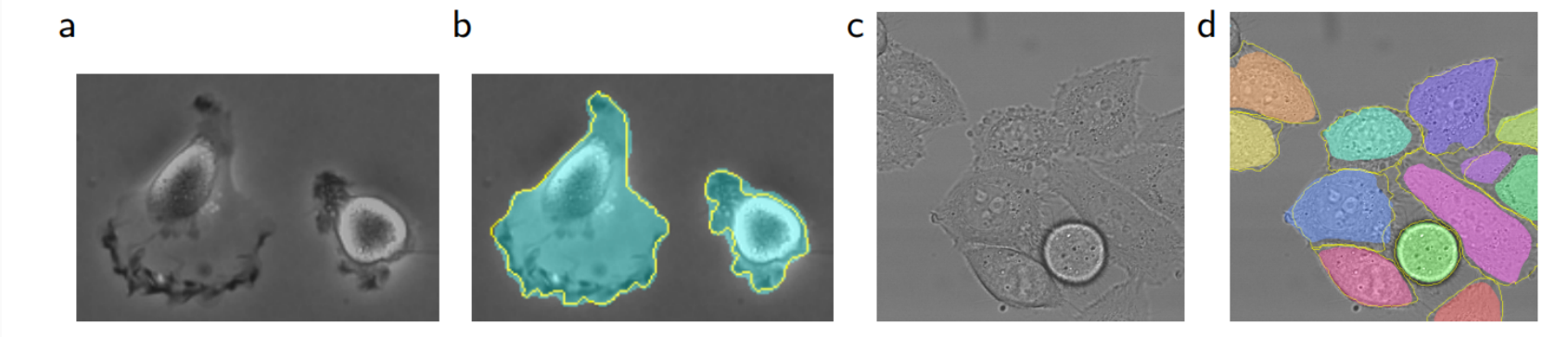
- Пиксели на границе предсказываем, например, отражая вход.

U-Net

- В исходной статье клетки сегментировали:

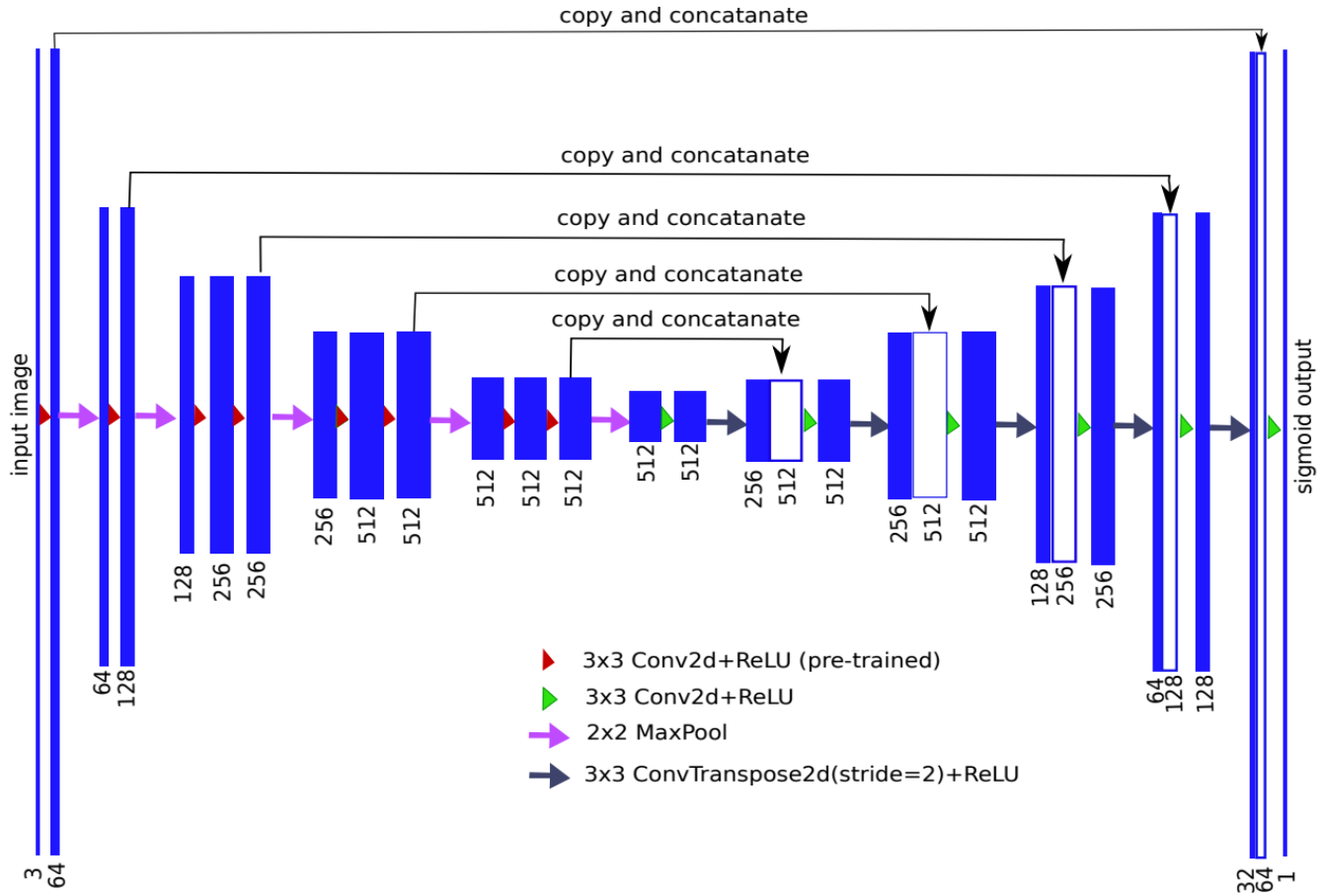


- Цвета – полученная маска, граница – ground truth:



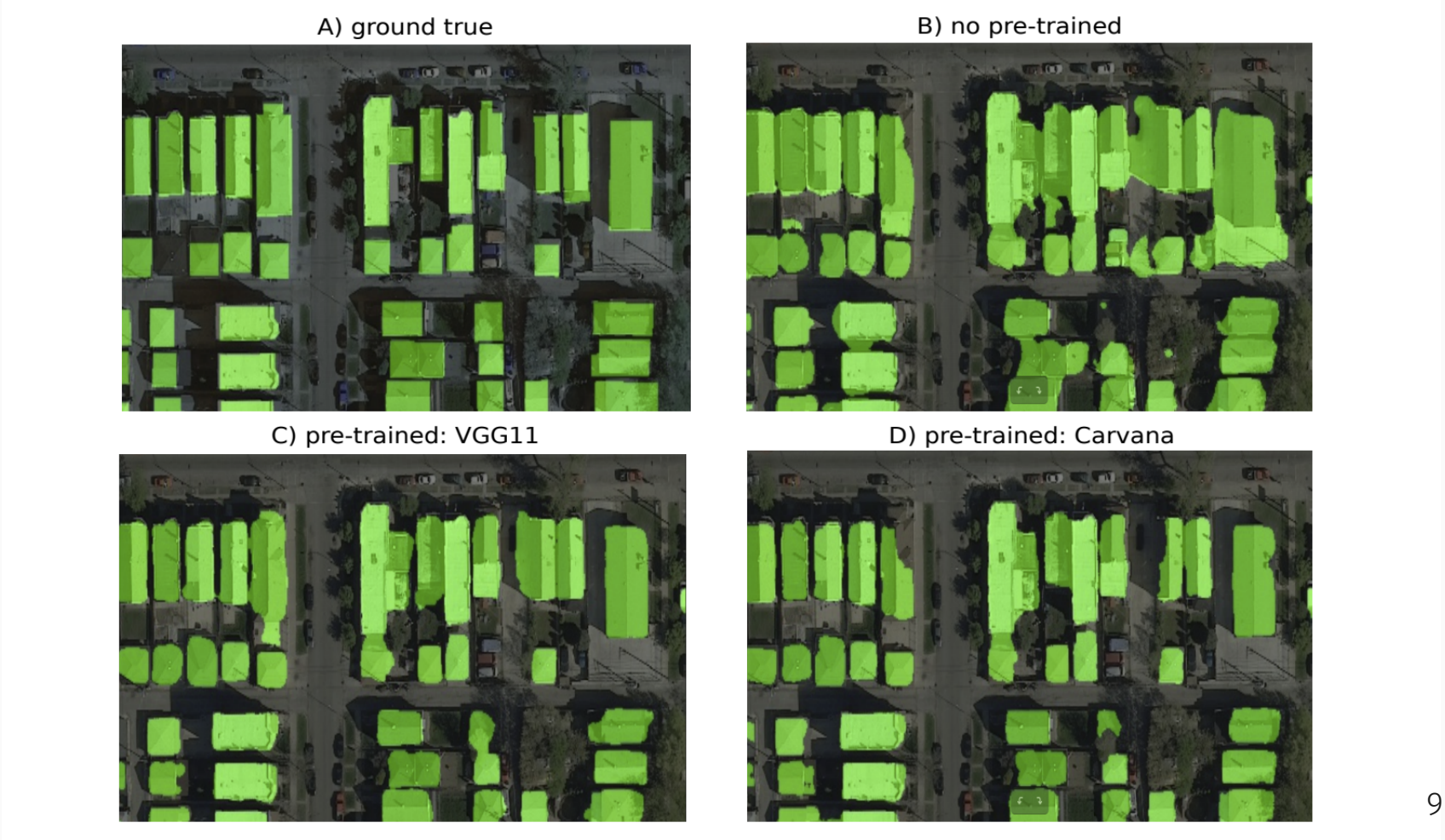
TernausNet и спутниковые снимки

- U-Net в последнее время стал основой для многих лучших моделей в сегментации с ограниченными датасетами.
- Вариант – TernausNet (Iglovikov, Shvets, 2018); основан на VGG11 без полносвязных слоёв:



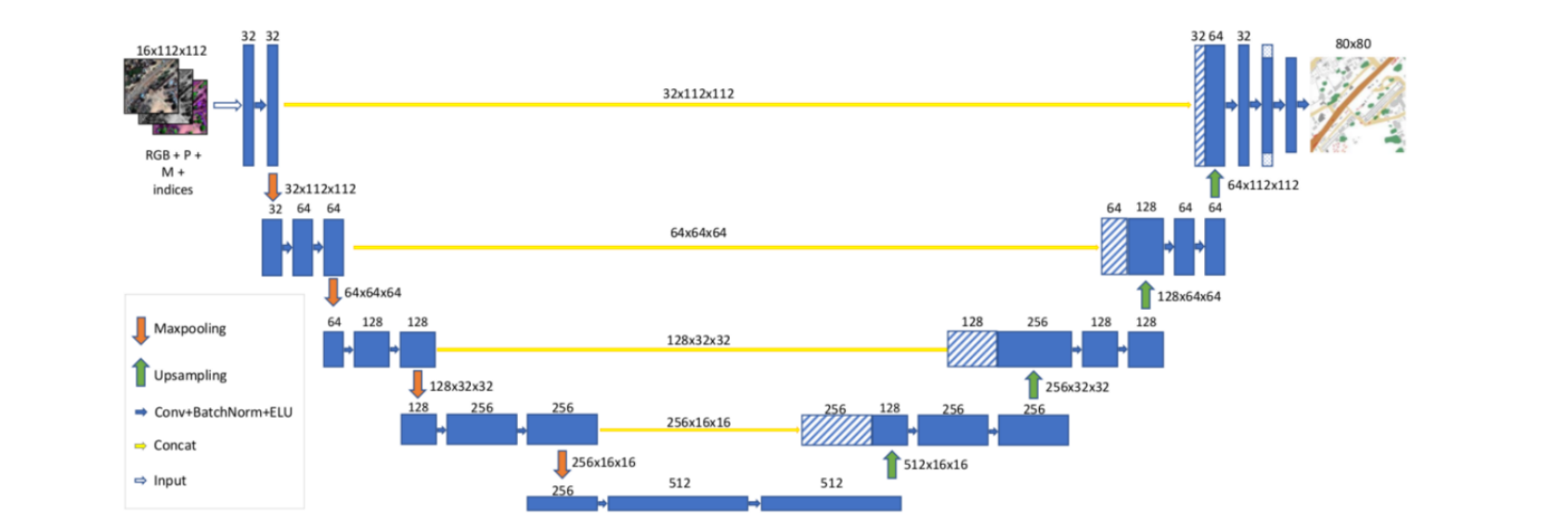
TernausNet и спутниковые снимки

- TernausNet выиграл Carvana Image Masking Challenge
- В (Iglovikov, Shvets, 2018) применён к сегментации спутниковых снимков:



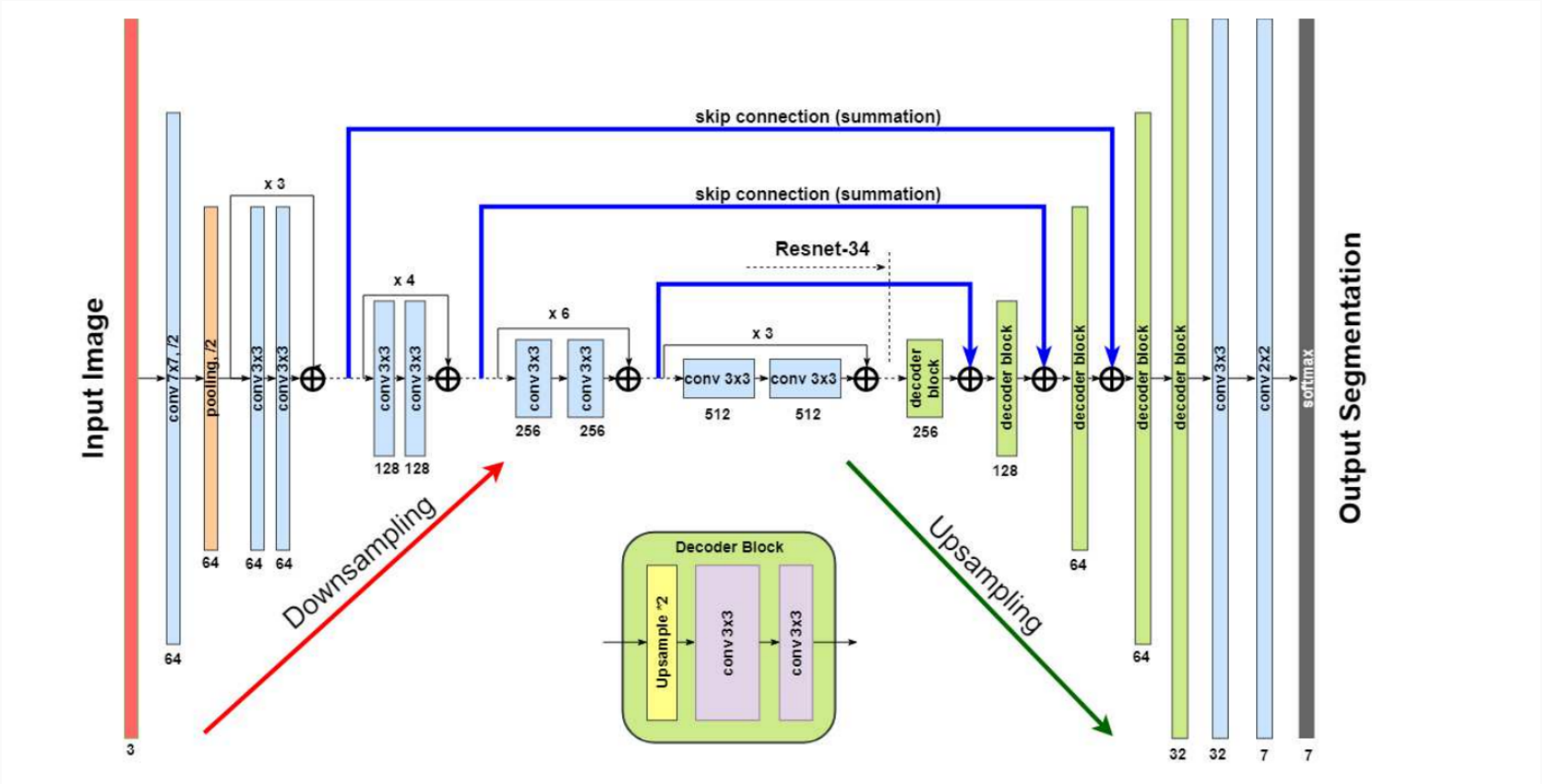
TernausNet и спутниковые снимки

- (Iglovikov et al., 2017): multispectral U-Net для Kaggle competition на спутниковых снимках:



TernausNet и спутниковые снимки

- (Rakhlin, Davydow, Nikolenko, 2018): U-Net для спутниковых снимков с ResNet-34



- Важный момент: какую функцию мы оптимизируем?

TernausNet и спутниковые снимки

- Обычно перекрёстную энтропию
- Но это не то, что нам нужно, нужно оптимизировать mIoU, т.е. средний коэффициент Жаккара:

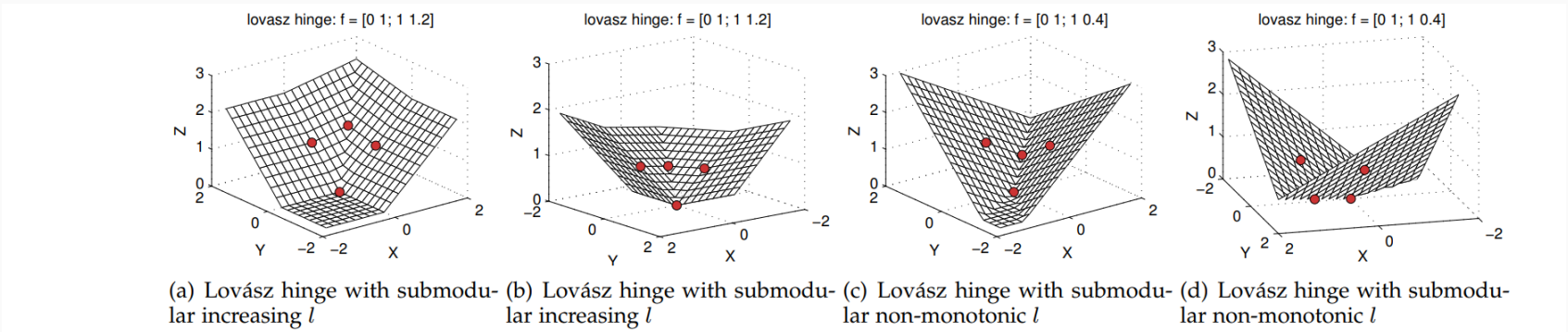
$$J_c(\hat{\mathbf{y}}, \mathbf{y}) = \frac{|\{\hat{\mathbf{y}} = c\} \cap \{\mathbf{y} = c\}|}{|\{\hat{\mathbf{y}} = c\} \cup \{\mathbf{y} = c\}|}.$$

- Но это, конечно, не продифференцируешь.
- Можно сделать мягкий вариант Жаккара:

$$J = \frac{1}{CN} \sum_c \sum_n \frac{y_n^c \hat{y}_n^c}{y_n^c + \hat{y}_n^c - y_n^c \hat{y}_n^c}.$$

TernausNet и спутниковые снимки

- Другая идея: давайте найдём выпуклую нижнюю оценку для Жаккара и её будем оптимизировать.
- (Yu, Blaschko, 2017) – Lovász hinge, нижняя оценка для субмодулярных функций:

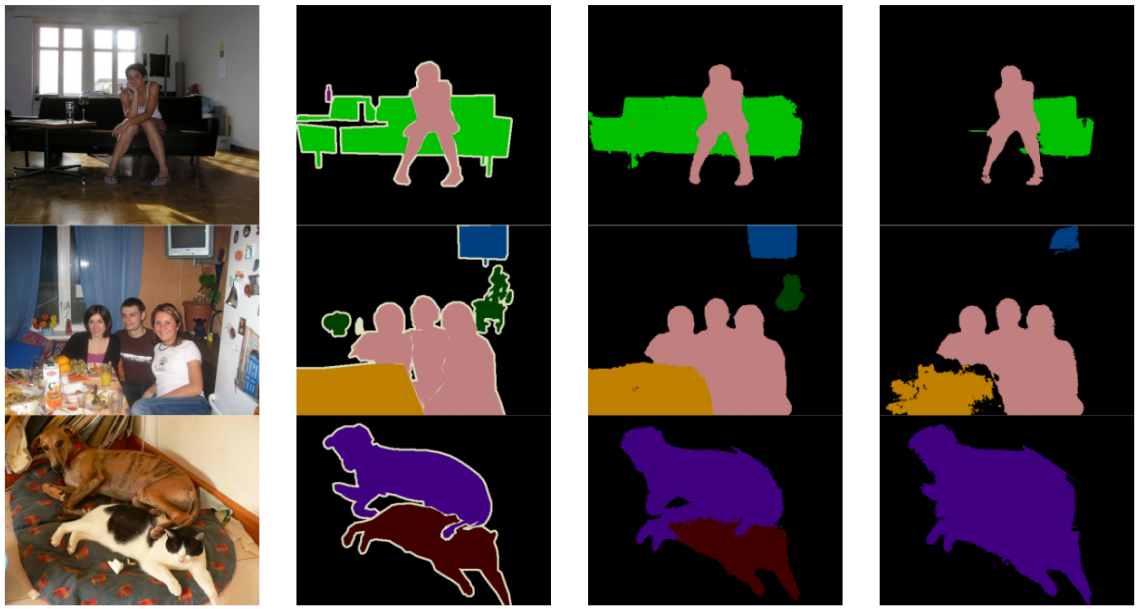


(a) Lovász hinge with submodular increasing l (b) Lovász hinge with submodular increasing l (c) Lovász hinge with submodular non-monotonic l (d) Lovász hinge with submodular non-monotonic l

- (Berman et al., 2018) – применили Lovász hinge для Жаккара с двумя классами и определили Lovász-Softmax loss, обобщение на несколько классов

TernausNet и спутниковые снимки

- (Berman et al., 2018) применили к сегментации:



(a) Input images (b) Ground truth masks (c) Lovász-Softmax + CRF (d) Cross-entropy + CRF

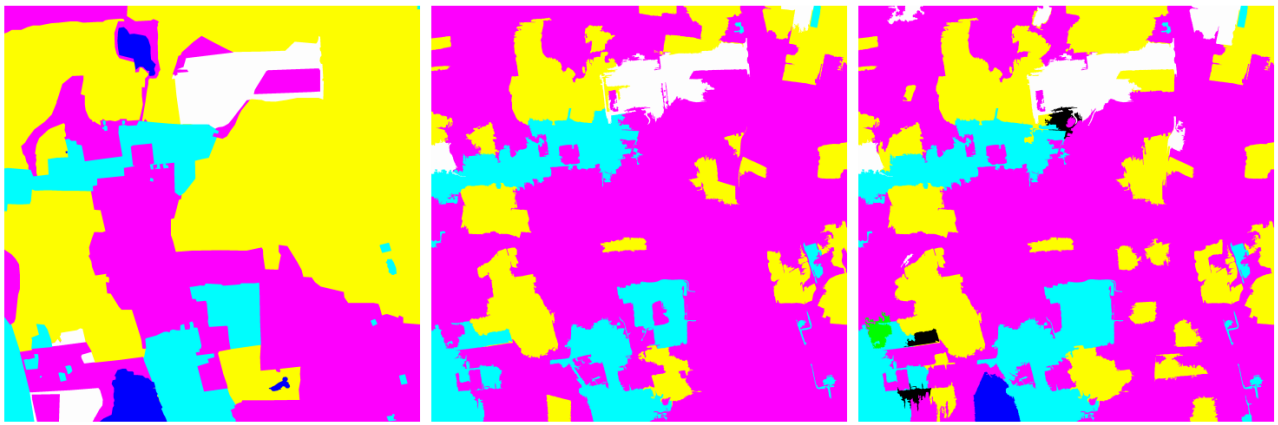
TernausNet и спутниковые снимки

- (Rakhlin, Davydow, Nikolenko, 2018): добавили Lovász-Softmax loss к U-Net, тоже хорошо стало



TernausNet и спутниковые снимки

- (Davydow, Nikolenko, 2018) – другая идея для оптимизации Жаккара: давайте сделаем postprocessing
- После сегментации мы знаем $p_x(c)$ для каждого суперпикселя x и класса c; можем оценить true positives и т.д.; давайте попробуем увеличивать ожидаемый Jaccard index, меняя значения суперпикселей по одному (суперпиксели ещё важны, если данные не слишком точные)



(a) (b) (c)

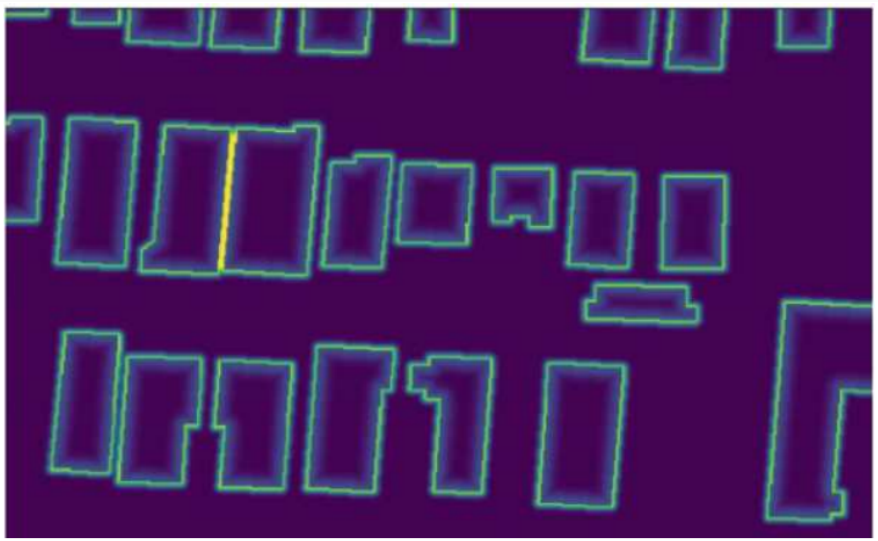
Figure 3. Sample segmentation results. Left to right: (a) ground truth mask; (b) model result before postprocessing, average Jaccard index 0.317, accuracy 0.572; (c) model result after postprocessing, average Jaccard index 0.393, accuracy 0.584.

TernausNet и спутниковые снимки

- (Golovanov et al., 2018) – как сегментировать здания на карте? Проблема в том, что границы трудно провести.
- Идея: давайте добавим веса в кросс-энтропию, которые будут зависеть от расстояния до ближайшей границы:

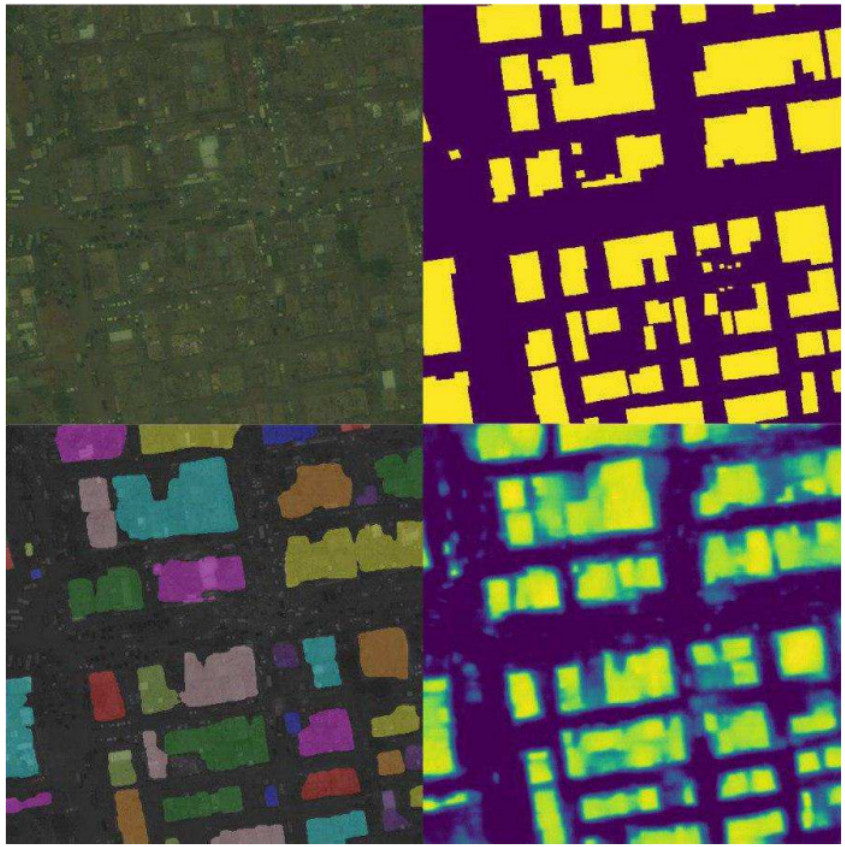
$$w(x) = w_c(x) + w_0 e^{-\frac{1}{2\sigma^2}(d_1(x)+d_2(x))^2} + w_0 \frac{d_3}{2r_d},$$

где d_1 – расстояние до ближайшего здания, d_2 – второго ближайшего, d_3 – до ближайшей узкой границы.



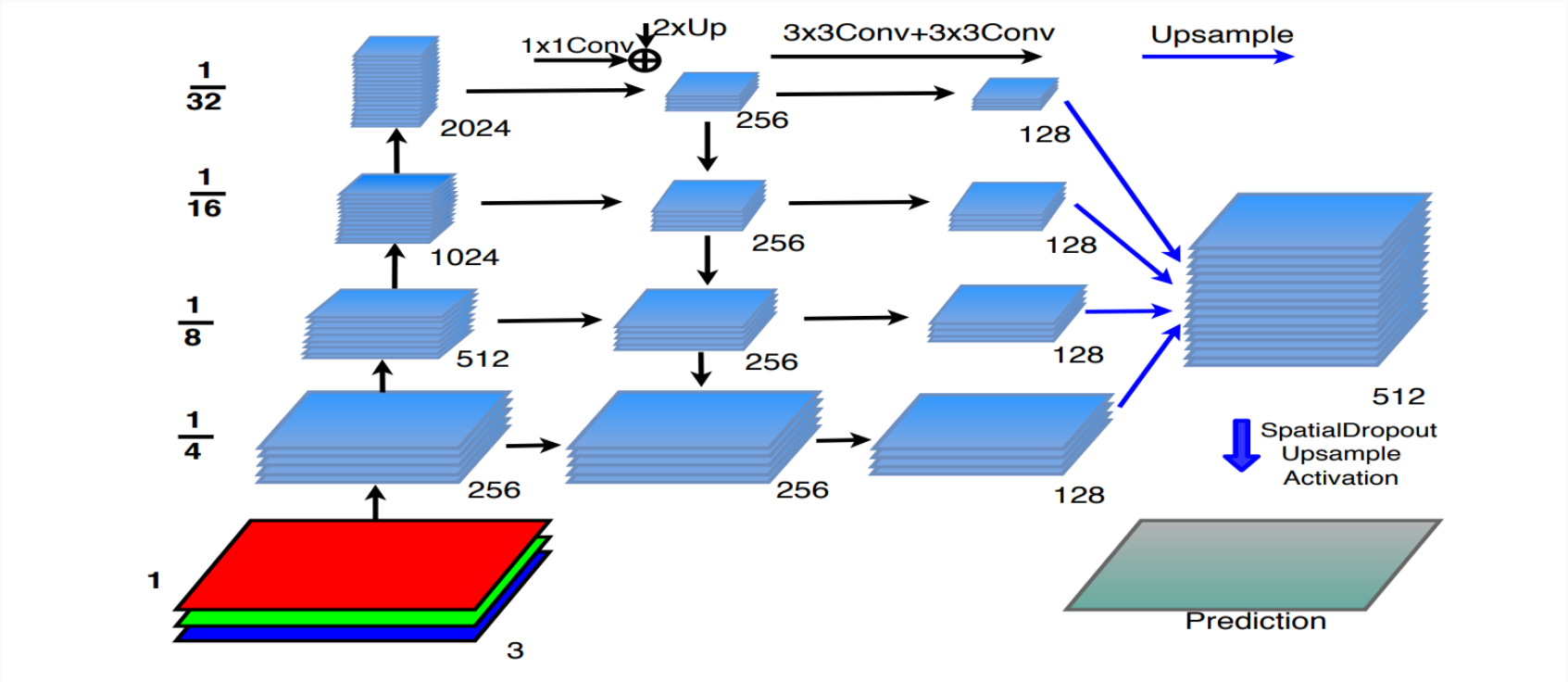
TernausNet и спутниковые снимки

- Помогает:



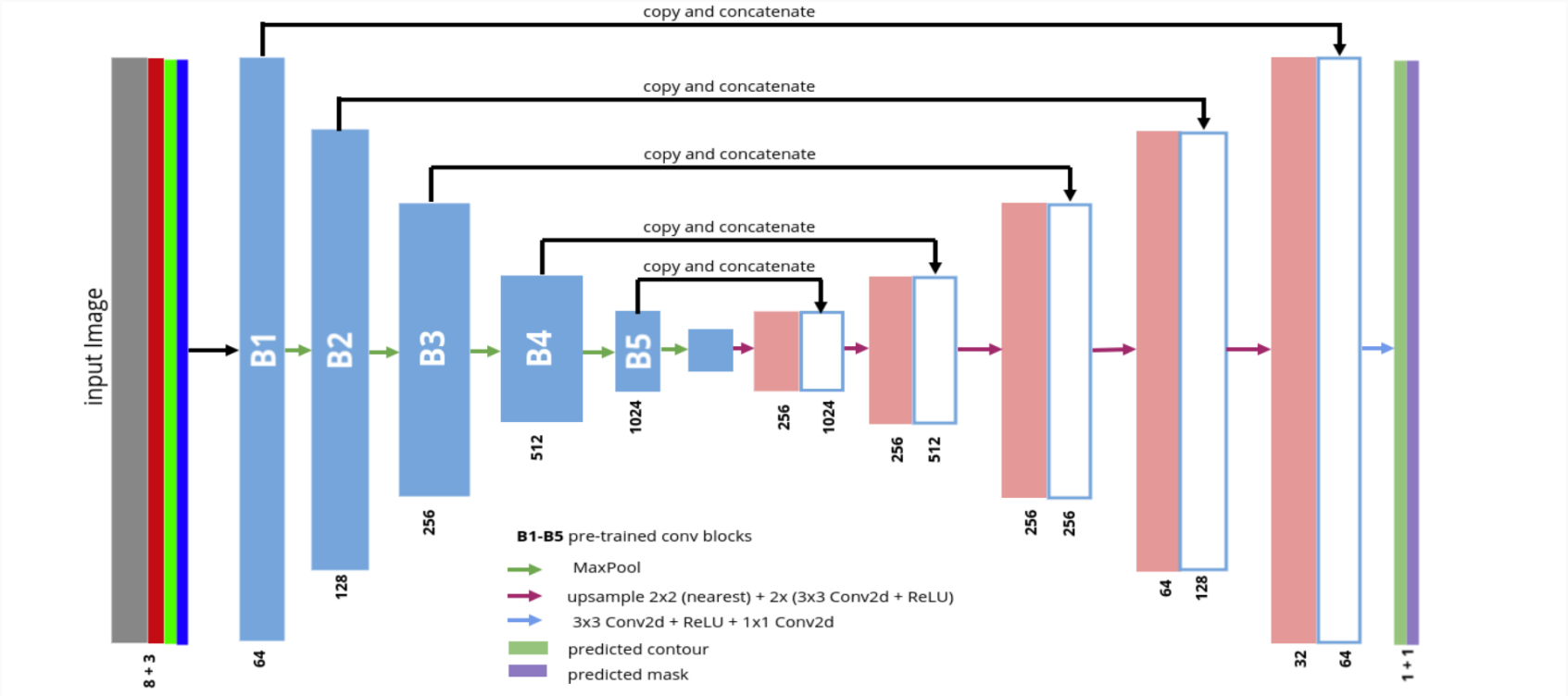
TernausNet и спутниковые снимки

- Кстати, в том конкурсе о зданиях всё тот же Игловиков адаптировал feature pyramid networks (Seferbekov et al., 2018):



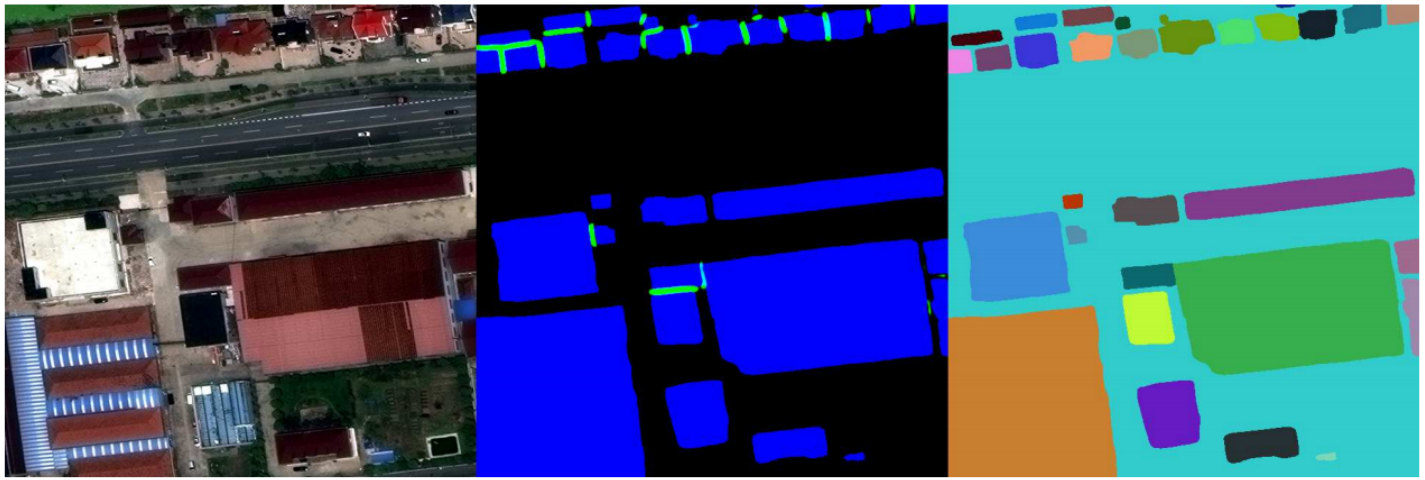
TernausNet и спутниковые снимки

- (Iglovikov et al., 2018): TernausNetV2; WideResNet-38 вместо VGG



TernausNet и спутниковые снимки

- Кроме того, RGB дополнено 11 multispectral channels, а на выходе есть отдельный канал, который предсказывает места, где объекты очень близки или касаются; это позволяет разделить здания и сделать instance segmentation:



TernausNet и спутниковые снимки

- Сейчас TernausNet и подобные архитектуры активно применяют в medical imaging.
- (Rakhlin et al., 2018): pediatric bone age assessment

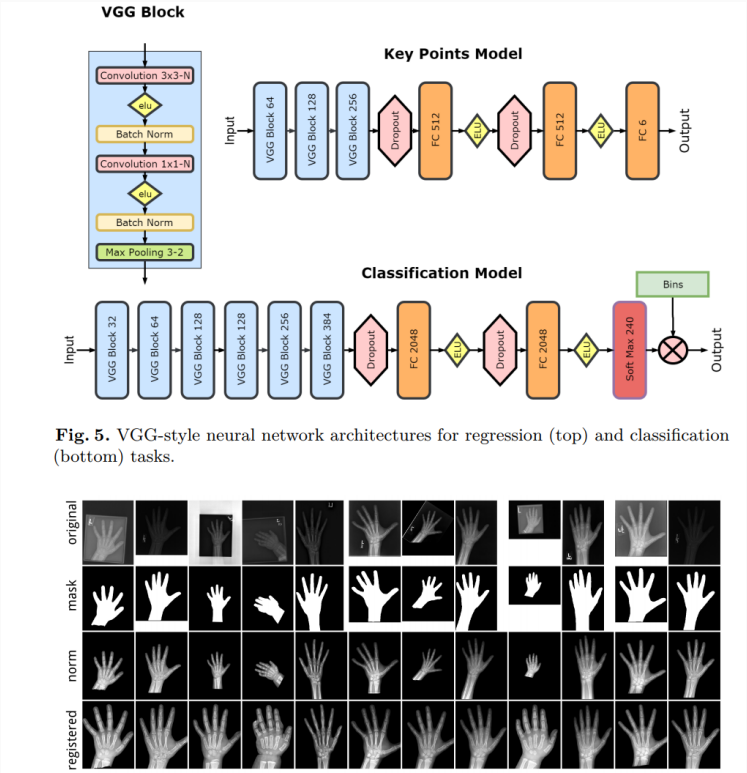
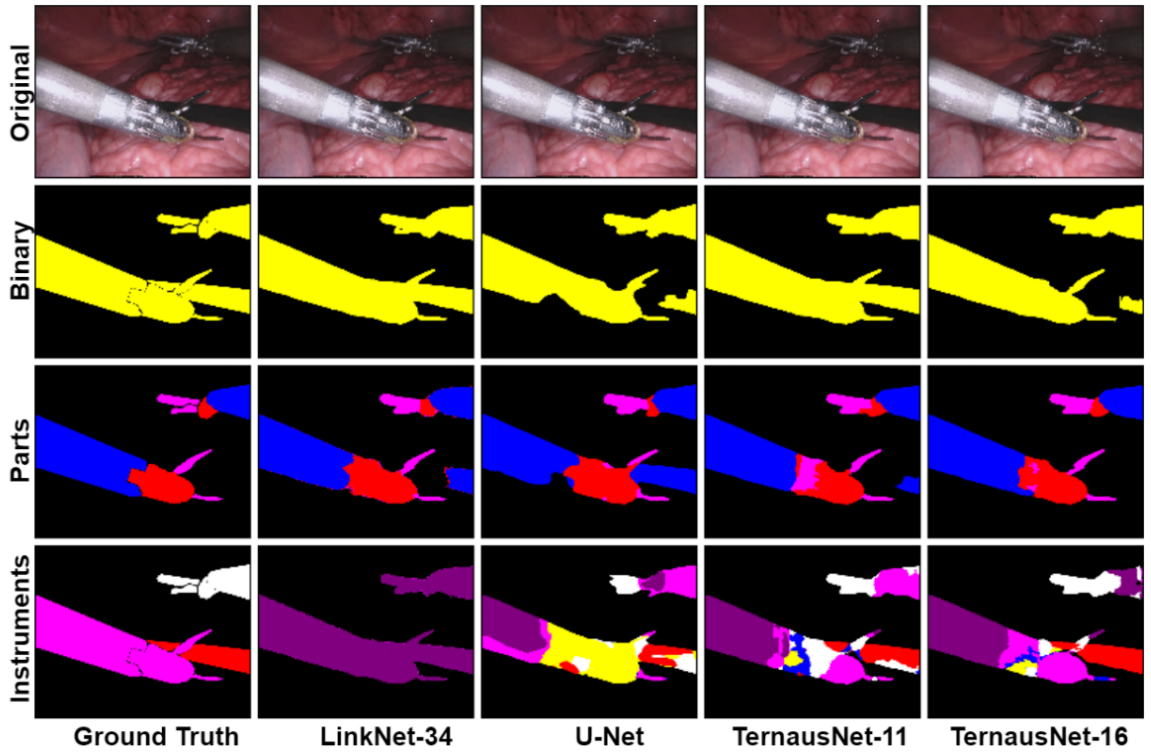


Fig. 5. VGG-style neural network architectures for regression (top) and classification (bottom) tasks.

TernausNet и спутниковые снимки

- (Shvets et al., 2018): сегментация инструментов для роботов-хирургов



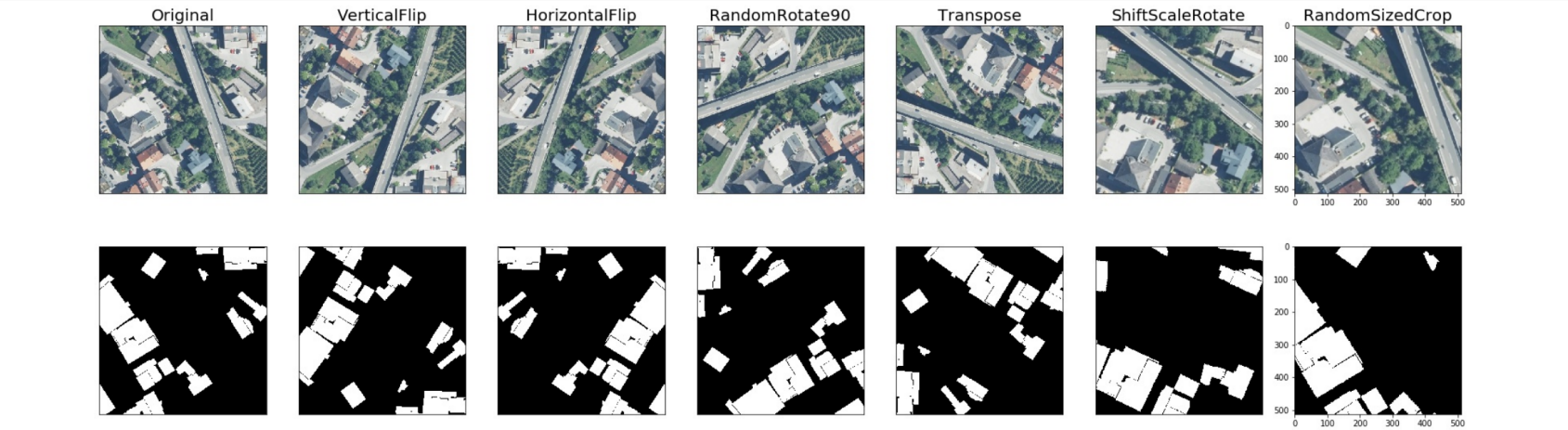
Albumentations

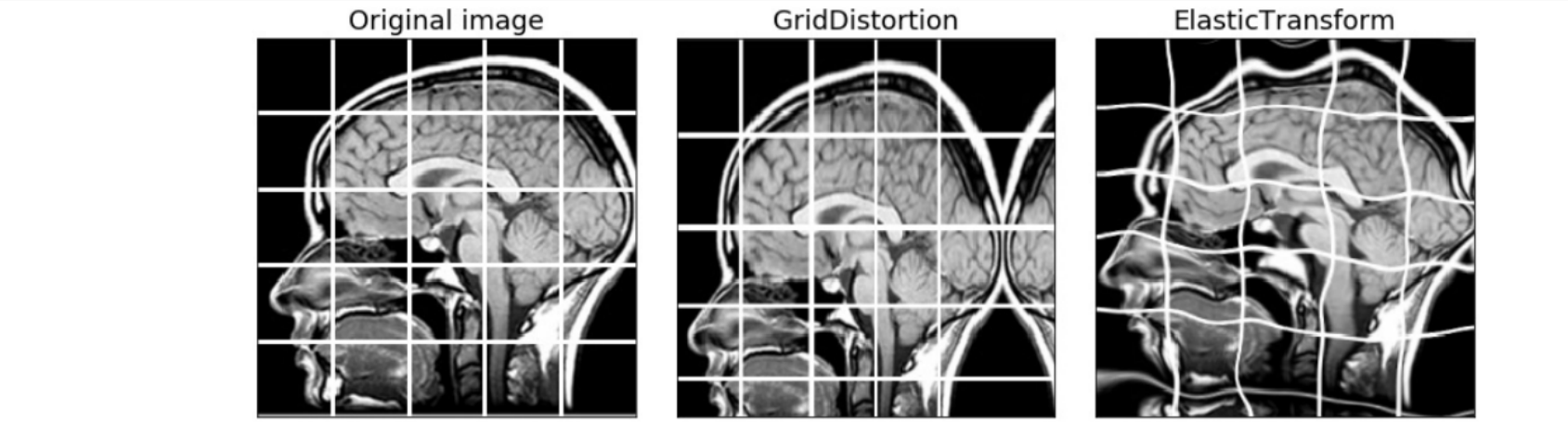
- Ну и раз мы об Игловикове, небольшое отступление
- Albumentations (Buslaev et al., 2018) – наверное, лучшая библиотека для data augmentation:



Albumentations

- Геометрические преобразования:





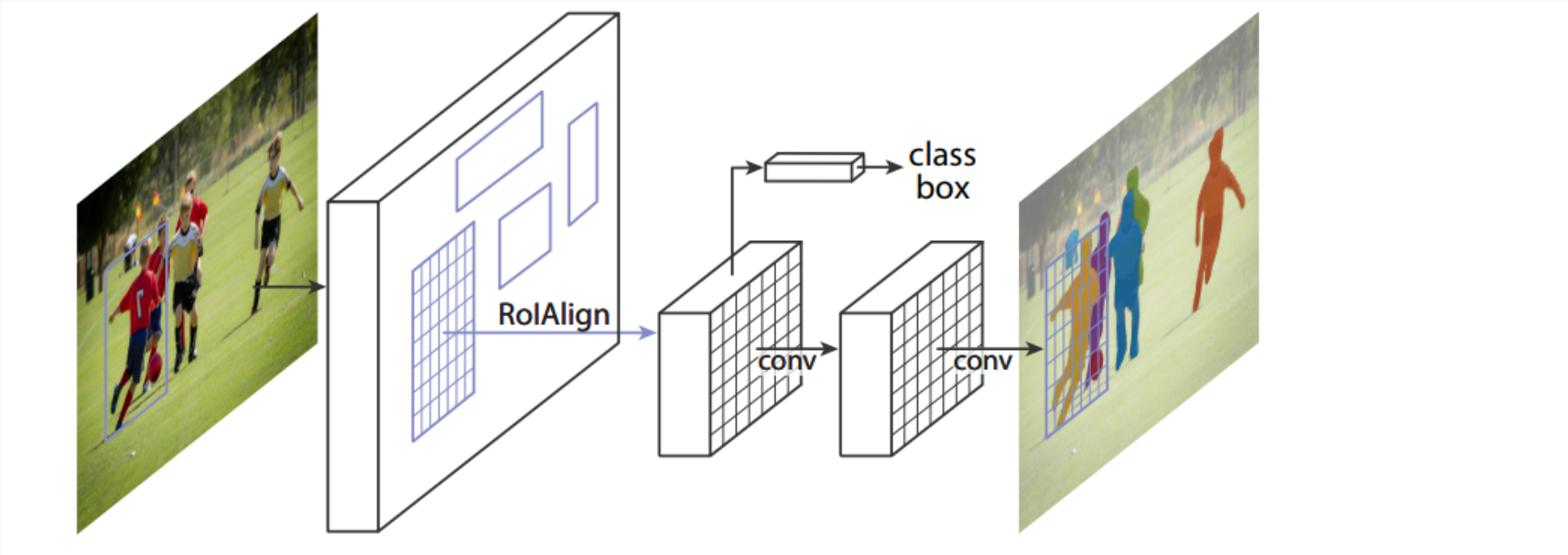
Albumentations

- Всё можно применить и к bbox'ам, и к маскам:



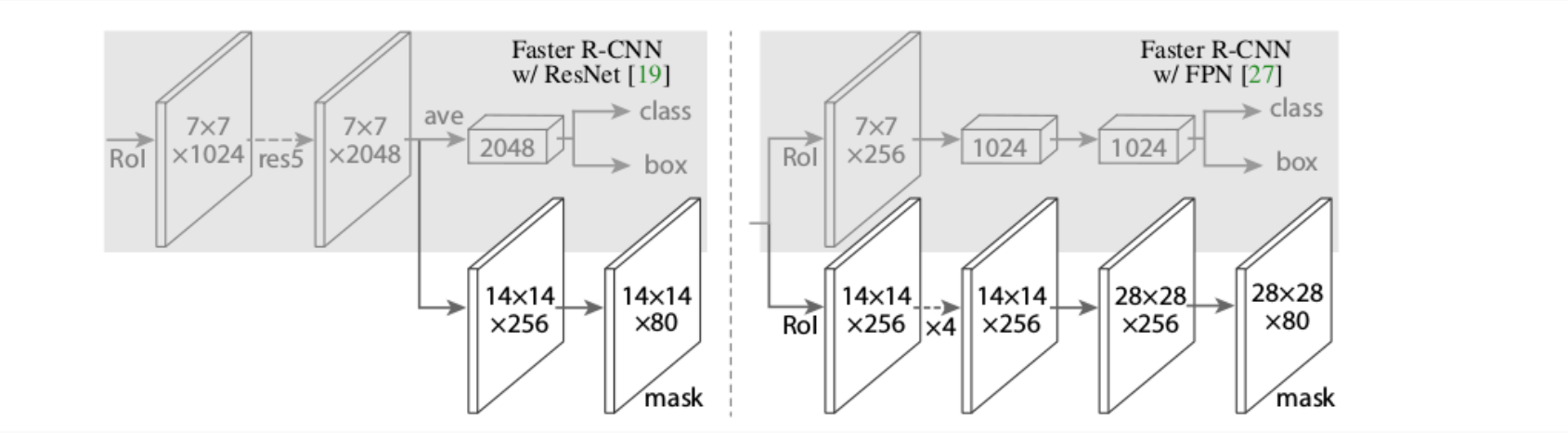
Mask R-CNN

- В прошлый раз мы много говорили о Faster R-CNN; оказывается, его можно адаптировать для сегментации:
  - у Faster R-CNN два выхода: метка класса и сдвиг bounding box;
  - а теперь давайте добавим третий выход, который считает прямо маску объекта, попиксельно.
- Получается *Mask R-CNN* (He et al., 2017).



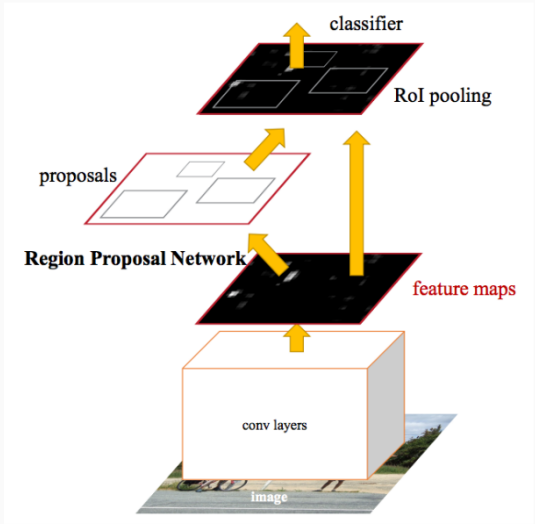
Mask R-CNN

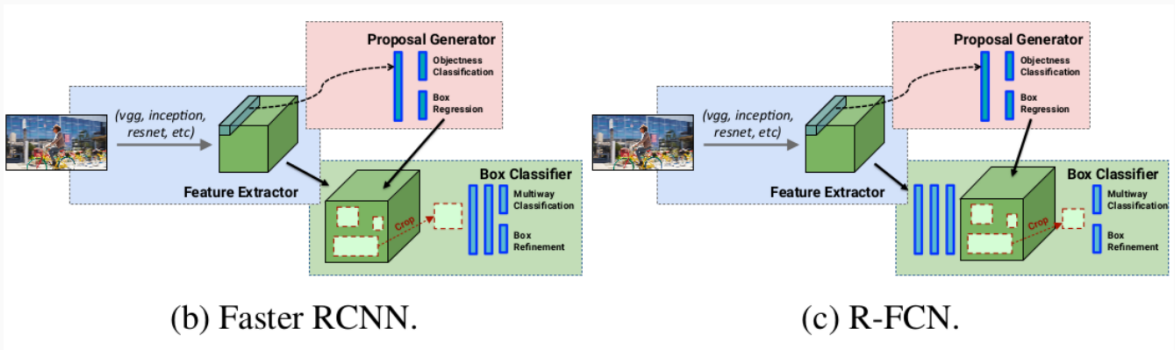
- Добавляем ошибку за маску (средняя кросс-энтропия по пикселям).
- Это можно добавить к разным базовым вариантам Faster R-CNN.



Mask R-CNN

- В принципе это очень прямолинейное расширение. Но есть тонкость: Faster R-CNN не приспособлен для точной локализации.





(b) Faster RCNN. (c) R-FCN.

Mask R-CNN

- RoIPool необратим и сильно уменьшает размер сетки
- В Mask R-CNN добавляется слой RoIAlign, который не делает квантование, а делает билинейную интерполяцию:

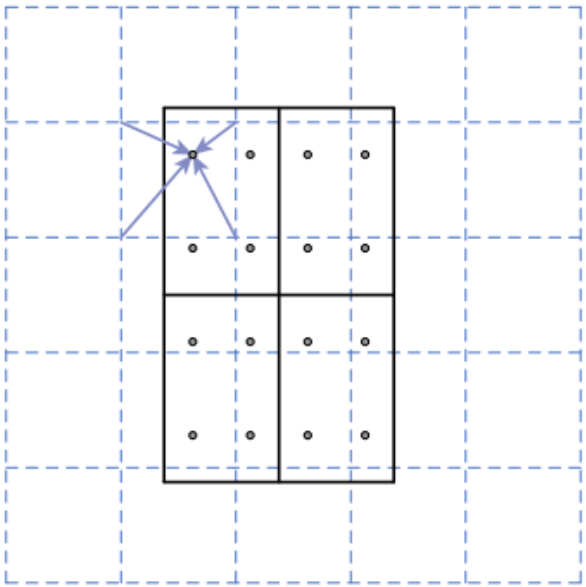
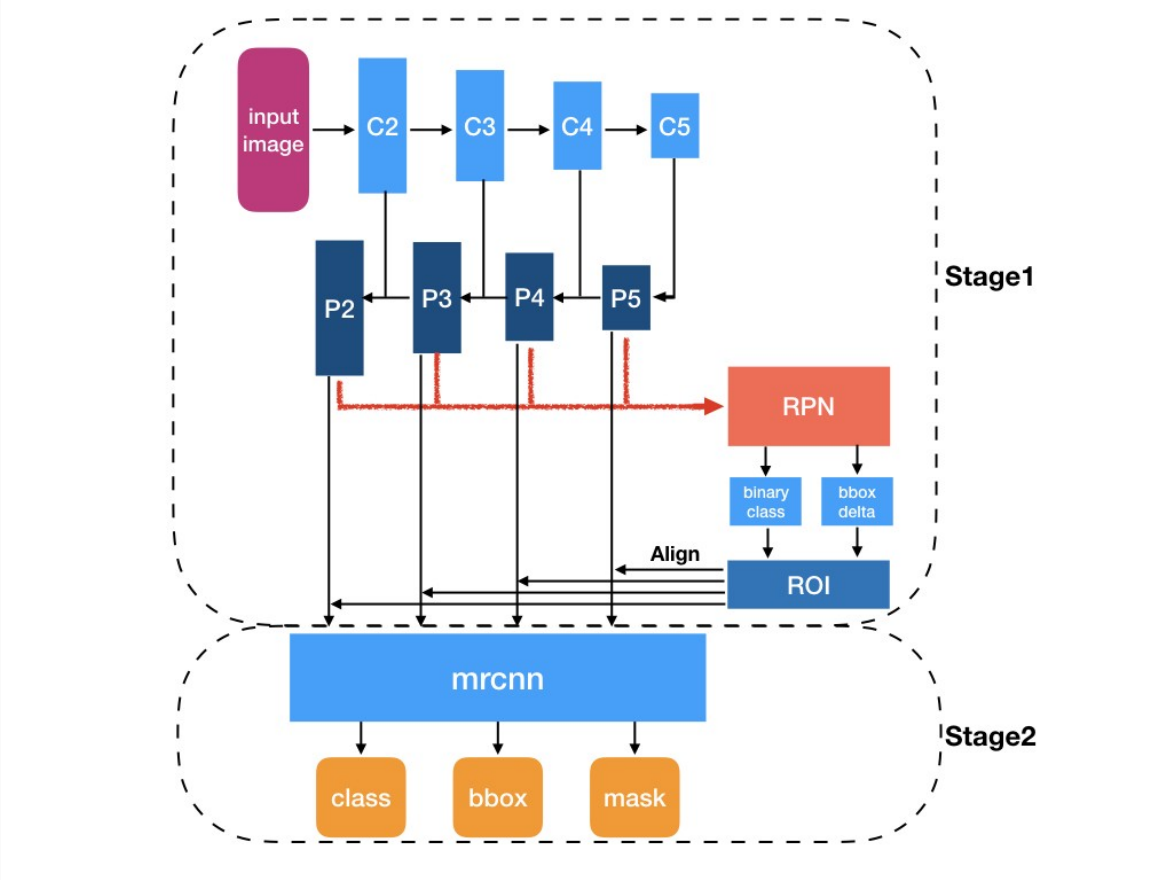


Figure 3. **RoIAlign:** The dashed grid represents a feature map, the solid lines an RoI (with 2×2 bins in this example), and the dots the 4 sampling points in each bin. RoIAlign computes the value of each sampling point by bilinear interpolation from the nearby grid points on the feature map. No quantization is performed on any coordinates involved in the RoI, its bins, or the sampling points.

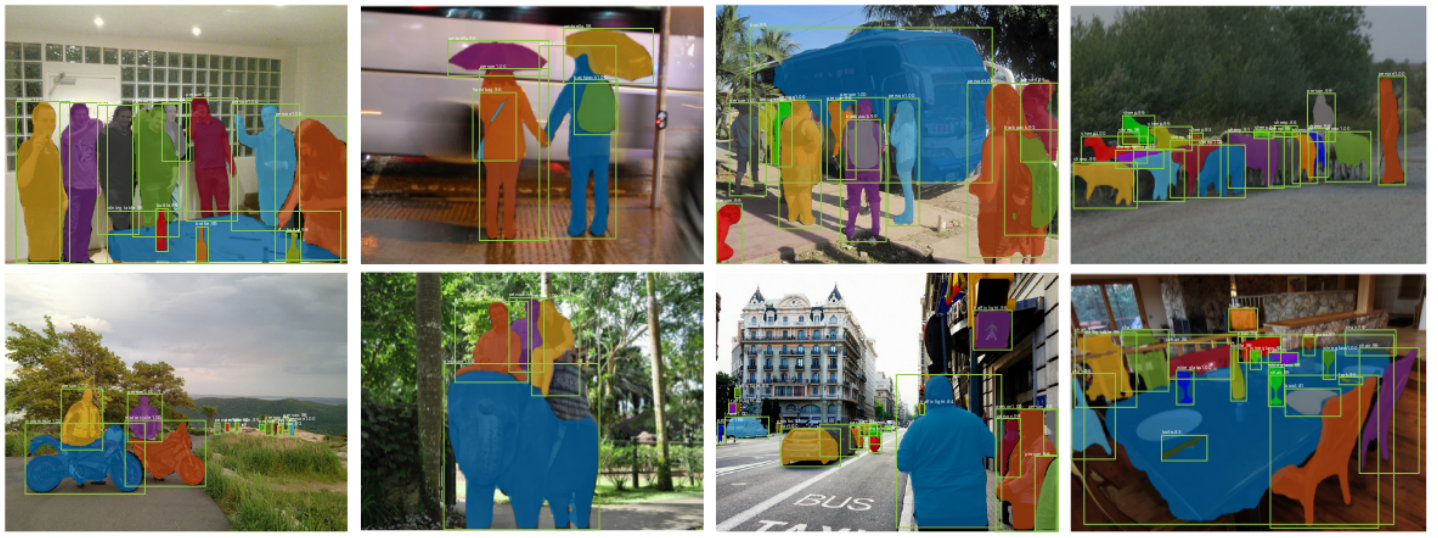
Mask R-CNN

- И его теперь можно вернуть на разные уровни признаков, чтобы распознавать объекты разных размеров:



Mask R-CNN

- Получается очень хорошо.





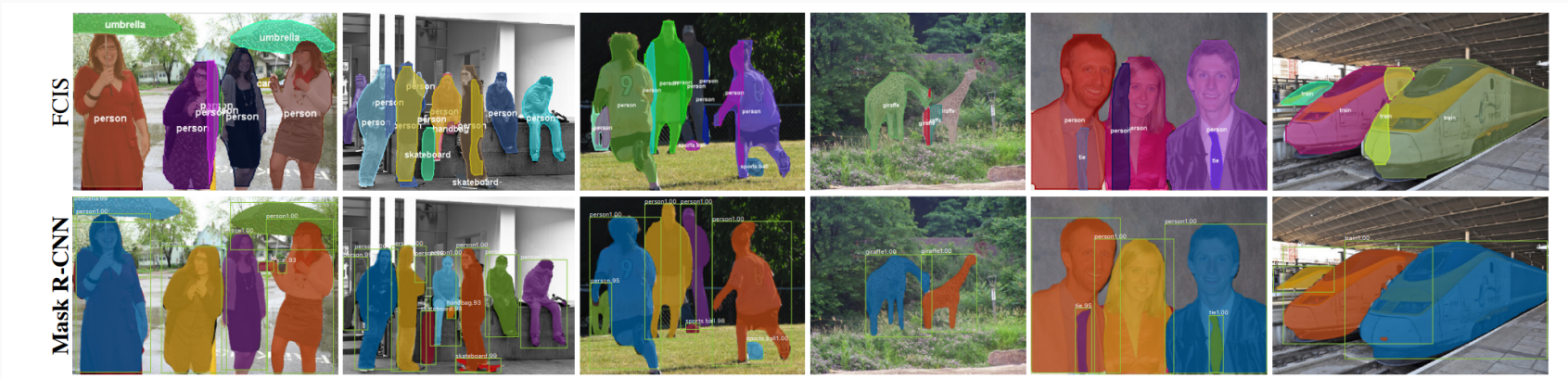
Mask R-CNN

- Ещё примеры из COCO:



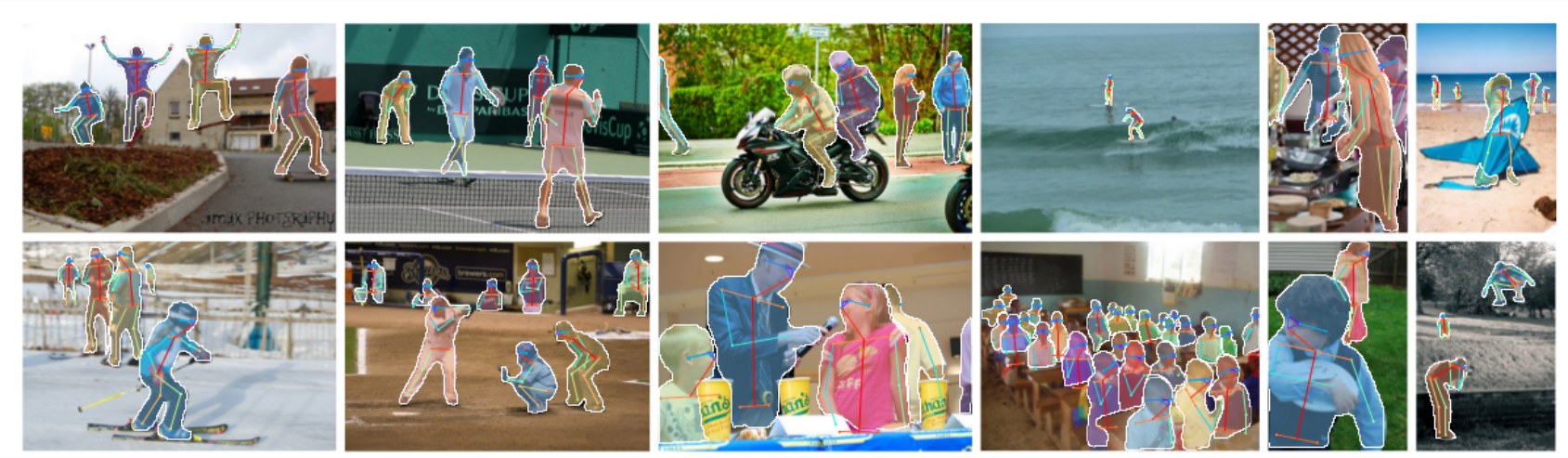
Mask R-CNN

- Причём очень хорошо работает именно instance segmentation, у конкурентов пересекающиеся объекты плохо обрабатываются:



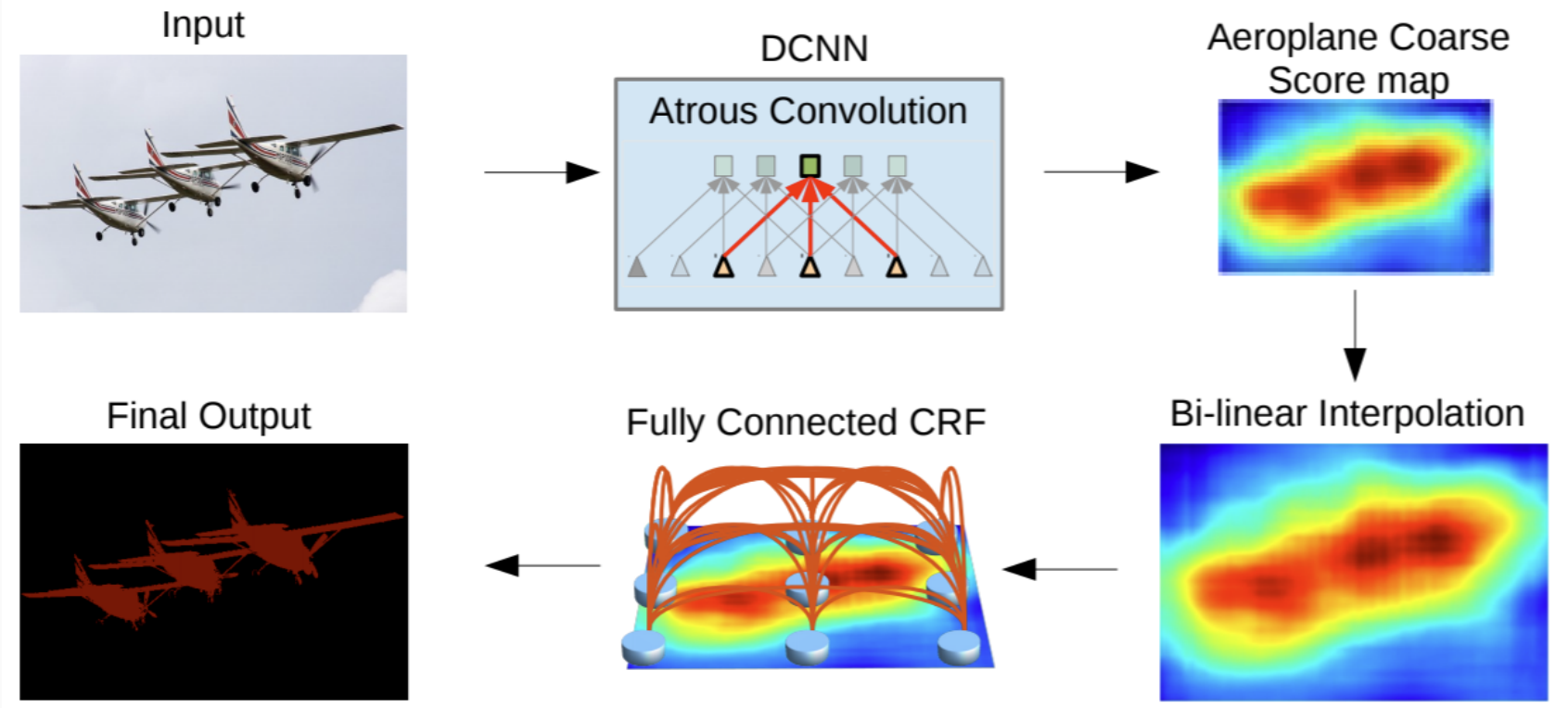
Mask R-CNN

- А можно, например, вместо маски распознавать ключевые точки позы человека.
- Гибкая архитектура, можно любые нужные пиксели пытаться вставить и начать распознавать.



DeepLab

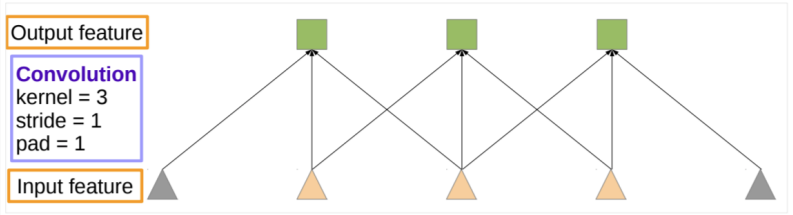
- DeepLab (Chen et al., 2016): разреженные свёртки + CRF



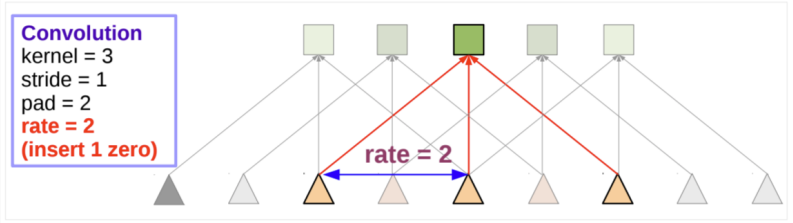
- CRF до сих пор остаются важной моделью (позже поговорим о них подробнее)

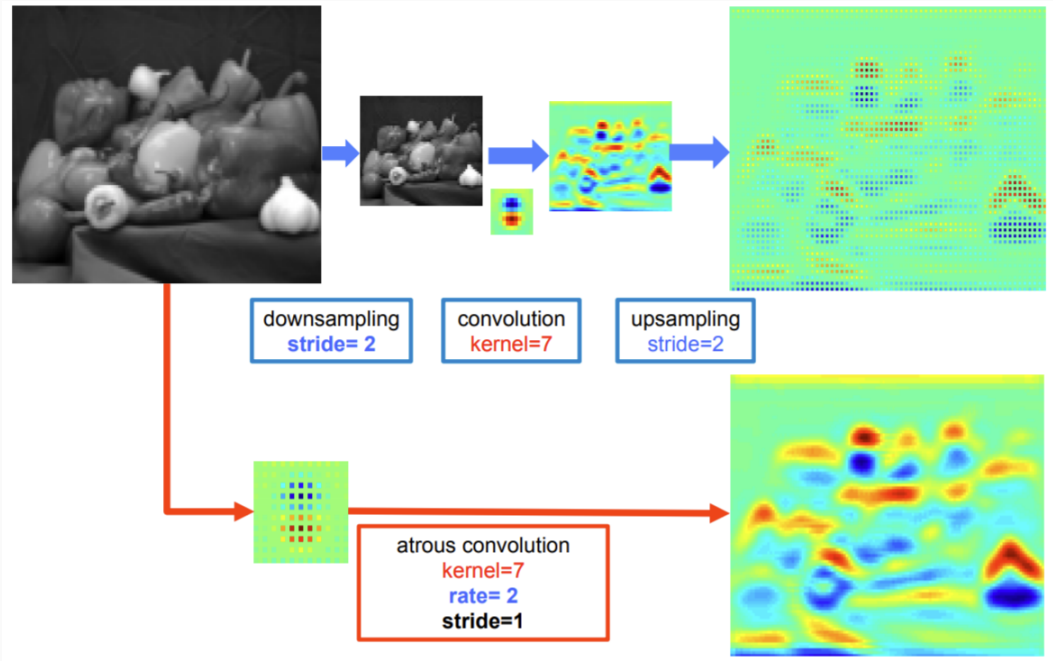
DeepLab

- Разреженные свёртки (atrous, or dilated convolutions):



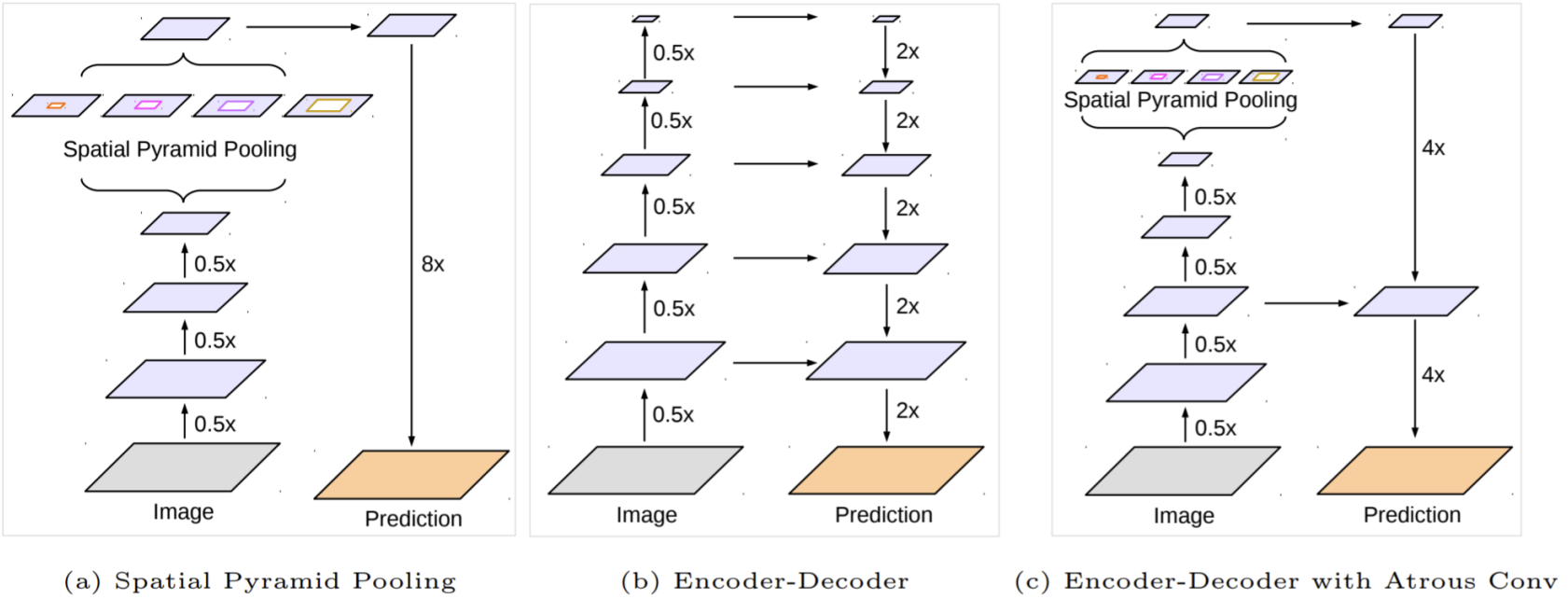
(a) Sparse feature extraction





DeepLab

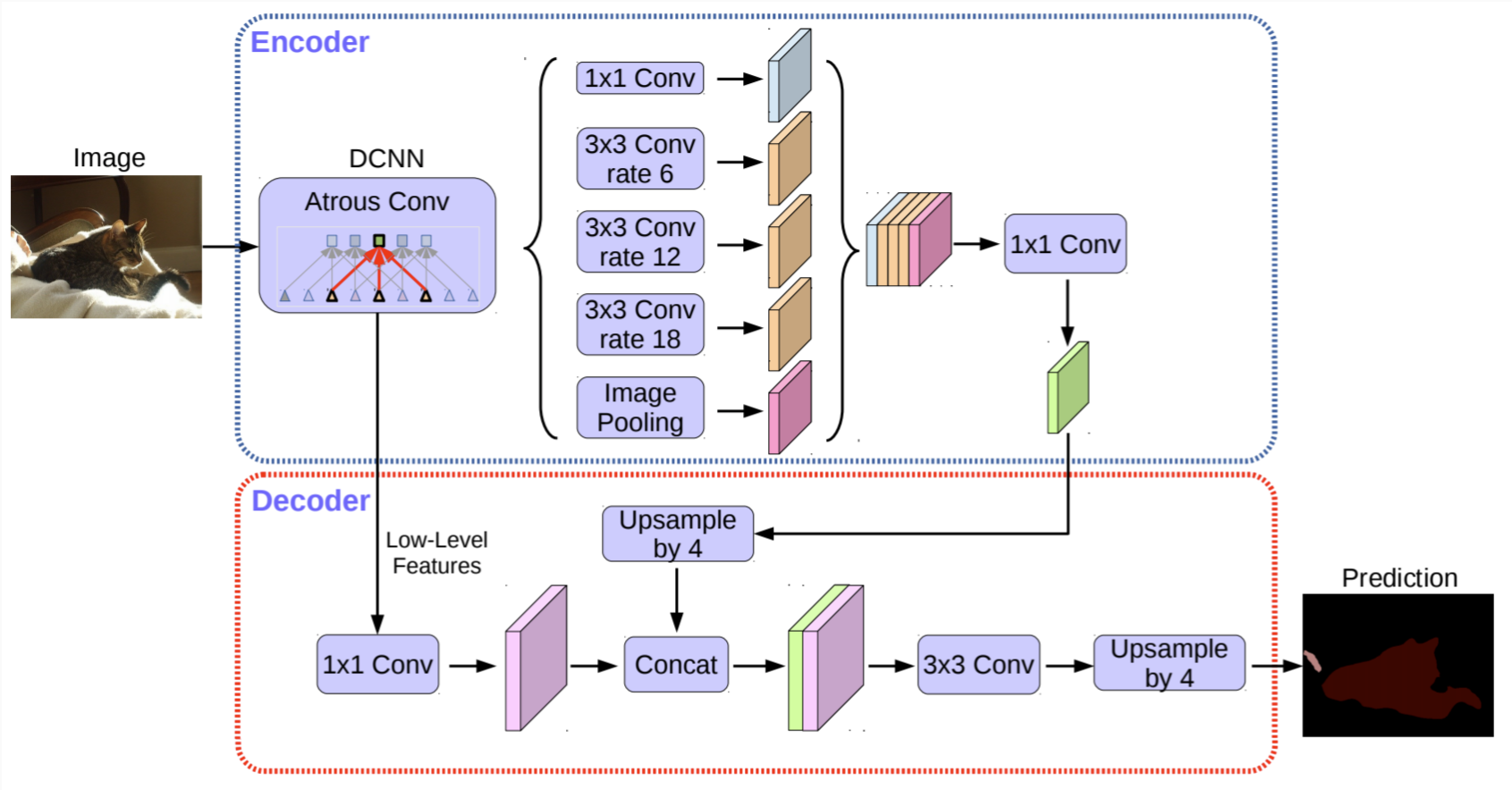
- DeepLab v3+ (Chen et al., 2018): от пирамидального пулинга к encoder-decoder



(a) Spatial Pyramid Pooling (b) Encoder-Decoder (c) Encoder-Decoder with Atrous Conv

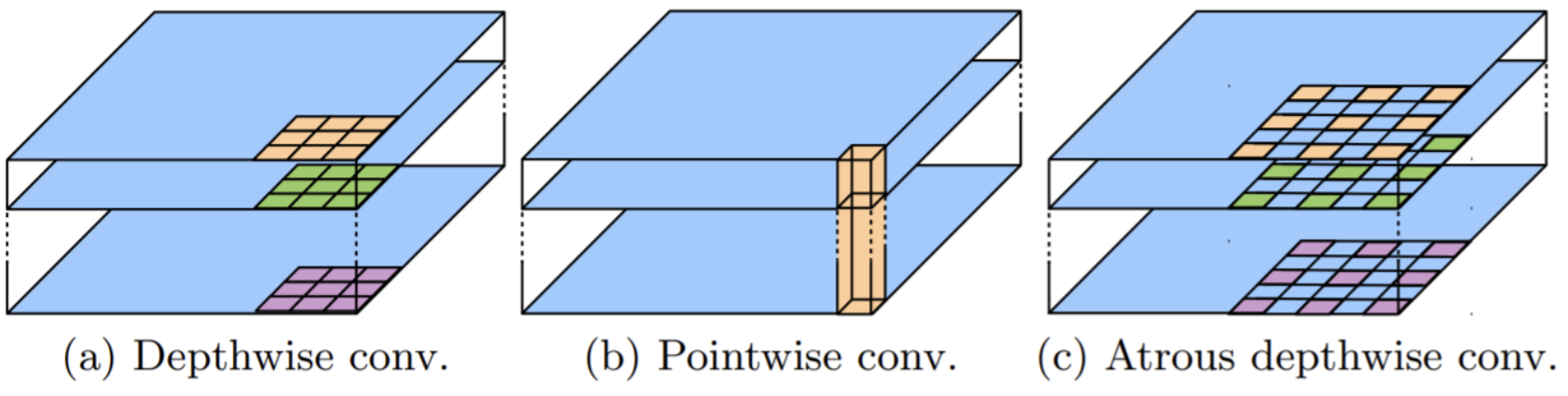
DeepLab

- DeepLab v3+ (Chen et al., 2018): архитектура



DeepLab

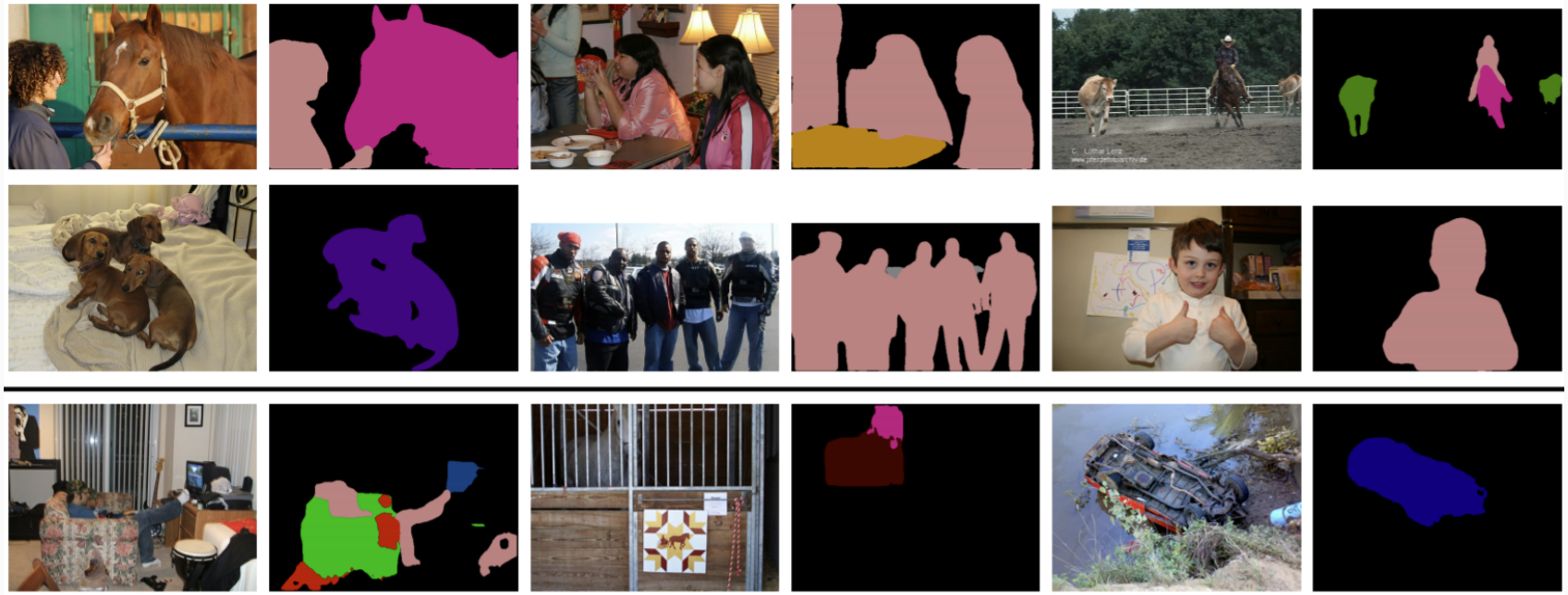
- DeepLab v3+ (Chen et al., 2018): depthwise separable convolutions



- Плюс модификации модуля Xception, плюс ещё куча маленьких улучшений...

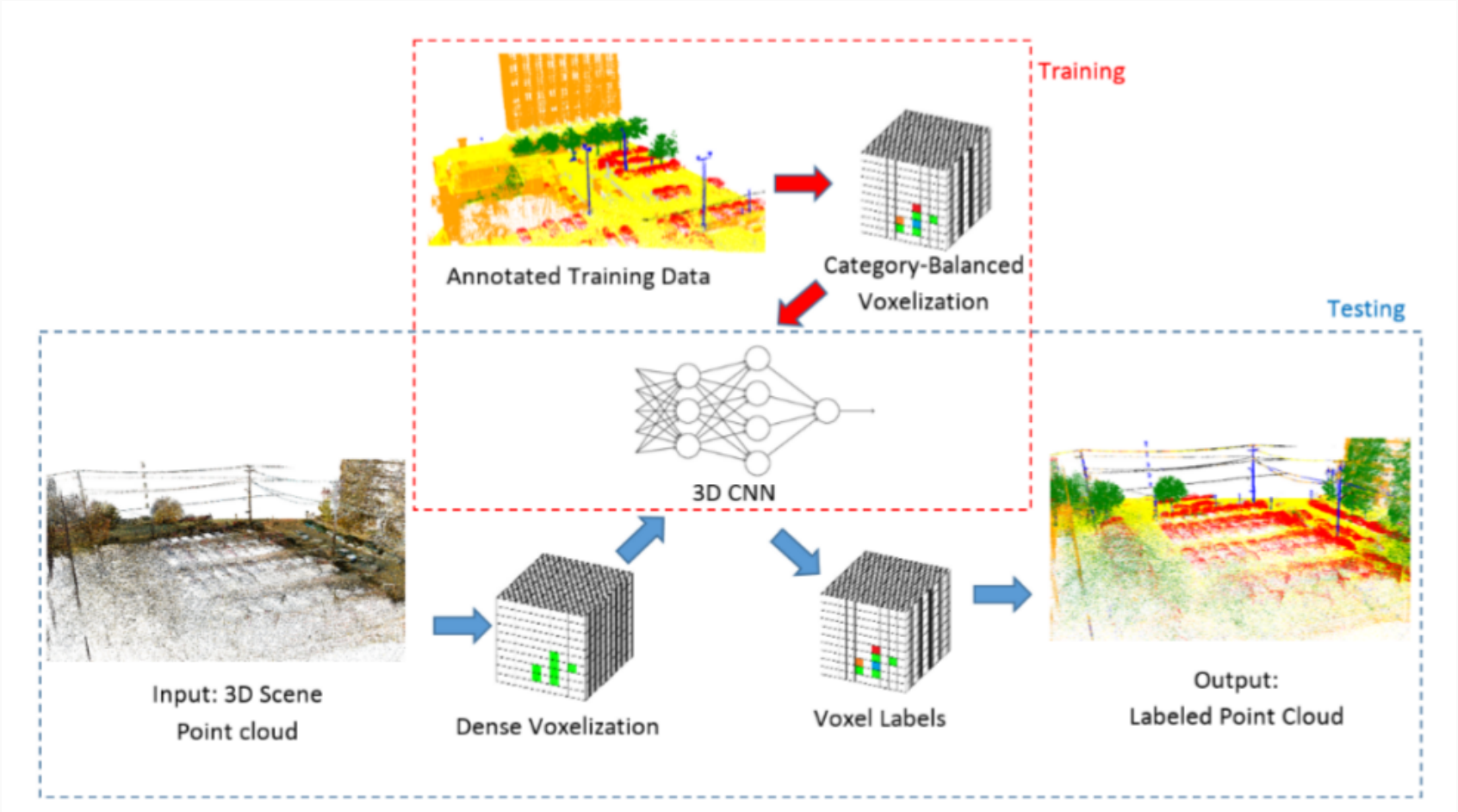
DeepLab

- DeepLab v3+ (Chen et al., 2018): хорошо работает и до сих пор составляет разумный baseline



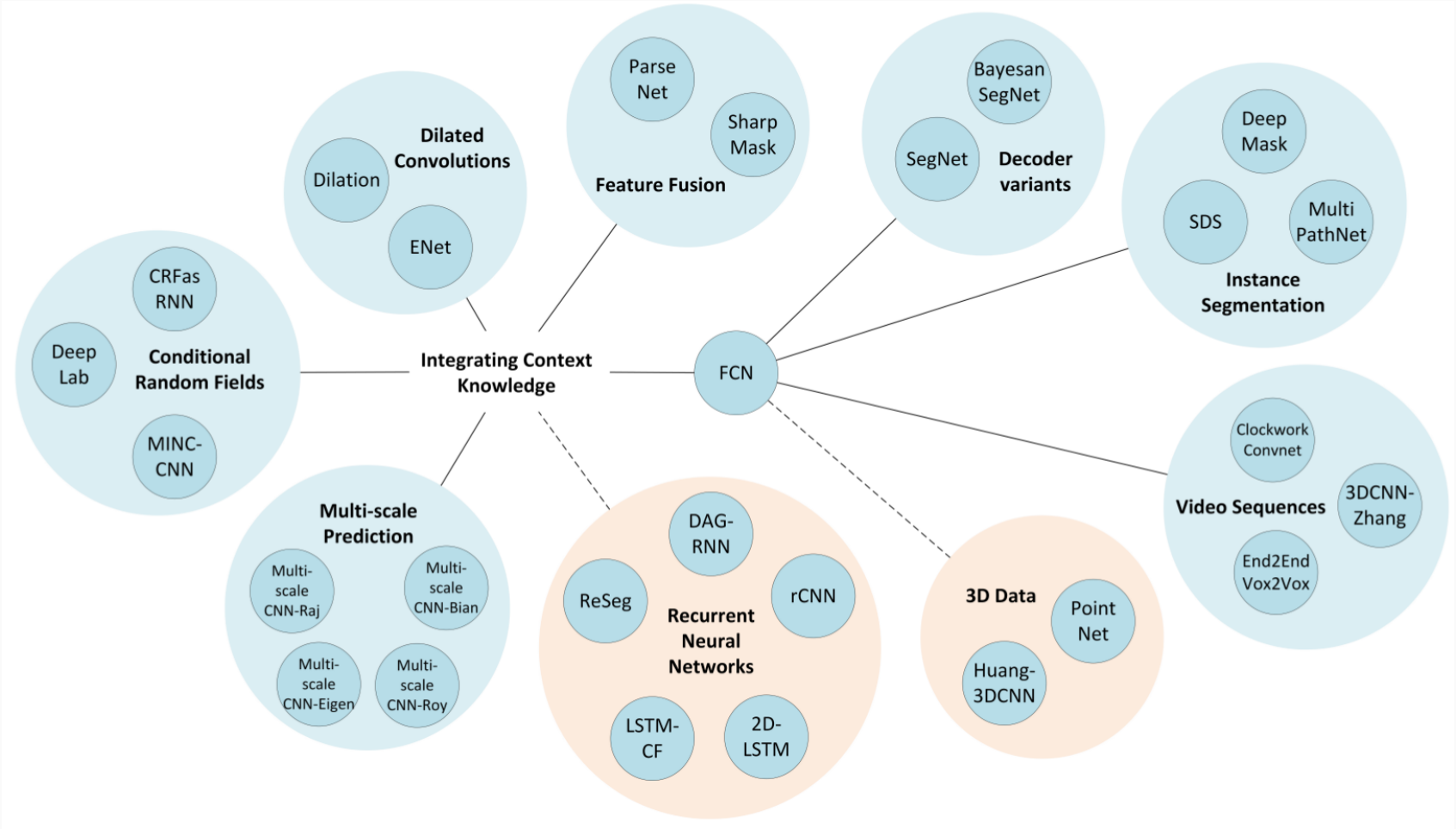
Сегментация в 3D

- А мы ещё совсем не коснулись сегментации в 3D:



Обзор

- И всё это всего лишь небольшая часть ландшафта сегментации, всего лишь примеры...

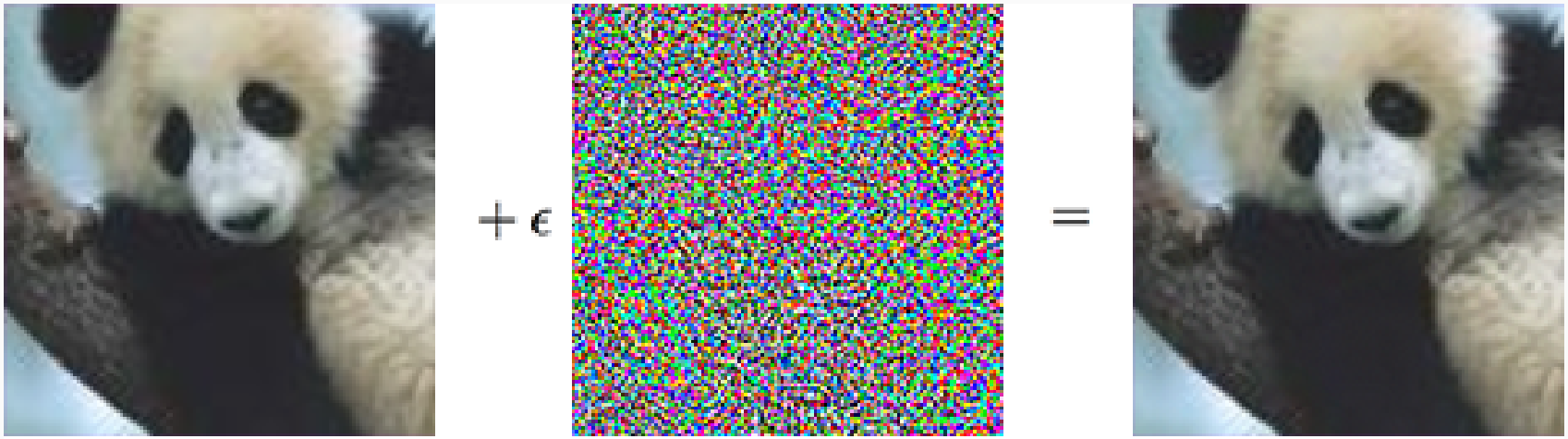


- А сейчас всё опять изменилось из-за трансформеров

Adversarial examples

Adversarial examples

- Интересная особенность глубоких сетей: можно обмануть любую сеть, сделать картинку подходящей под любой класс неразличимым на человеческий взгляд шумом.



- Как?..

Adversarial examples

- Давайте сделаем градиентный спуск не по весам сети θ, а по входу $\mathbf{x}$!
- Надо только контролировать, чтобы новый пример $\hat{\mathbf{x}}$ оставался похож на исходный $\mathbf{x}$, например чтобы $\|\hat{\mathbf{x}} - \mathbf{x}\|_\infty \leq \epsilon$.
- Более того, можно попробовать сделать $\hat{\mathbf{x}}$ устойчивым ко всяким преобразованиям вроде поворотов.
- Кстати, а это как сделать?
- Давайте посмотрим на пример...

Adversarial examples

- Направление началось в *Intriguing properties of neural networks* (Szegedy et al., 2013). Вообще очень интересная статья...
- Например, мы с вами анализировали “значения” нейронов, находя сильнейшие их активации.
- Т.е. предполагается, что если мы проанализируем нейроны последнего уровня, то это будет правильный базис в латентном пространстве, на котором легко выделить семантику.
- Правильно?..

Adversarial examples

- ...не совсем:



(a) Unit sensitive to lower round stroke.



(b) Unit sensitive to upper round stroke, or lower straight stroke.



(c) Unit senstive to left, upper round stroke.



(d) Unit senstive to diagonal straight stroke.

Figure 1: An MNIST experiment. The figure shows images that maximize the activation of various units (maximum stimulation in the natural basis direction). Images within each row share semantic properties.



(a) Direction sensitive to upper straight stroke, or lower round stroke.



(b) Direction sensitive to lower left loop.



(c) Direction senstive to round top stroke.



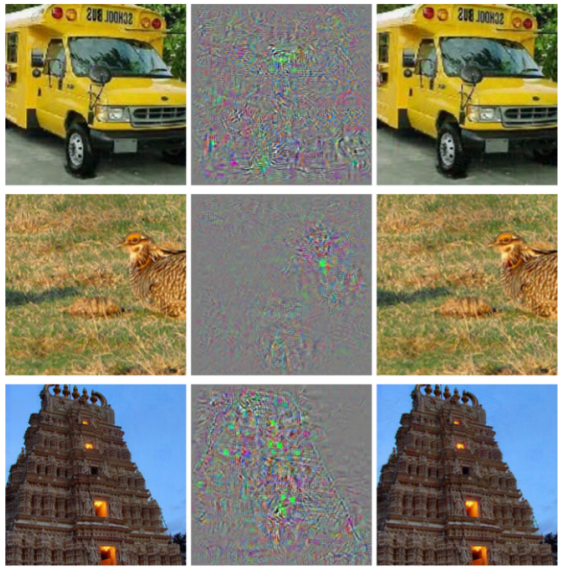
(d) Direction sensitive to right, upper round stroke.

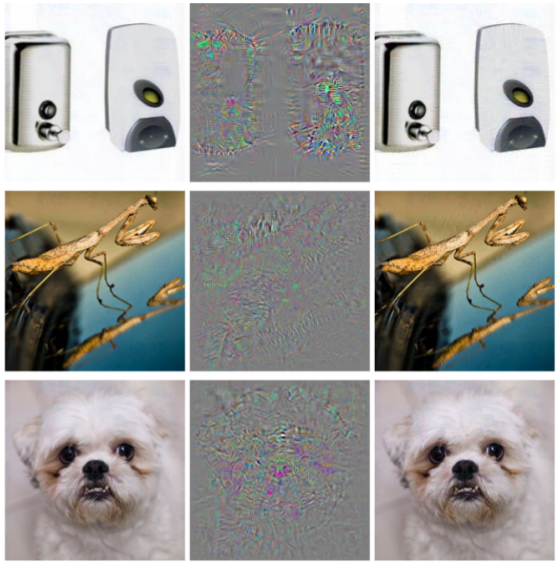
Figure 2: An MNIST experiment. The figure shows images that maximize the activations in a random direction (maximum stimulation in a random basis). Images within each row share semantic properties.

- T.e. у обычных сетей никакого нет disentanglement, пространство признаков хорошее, но базис в нём не лучше случайного.

Adversarial examples

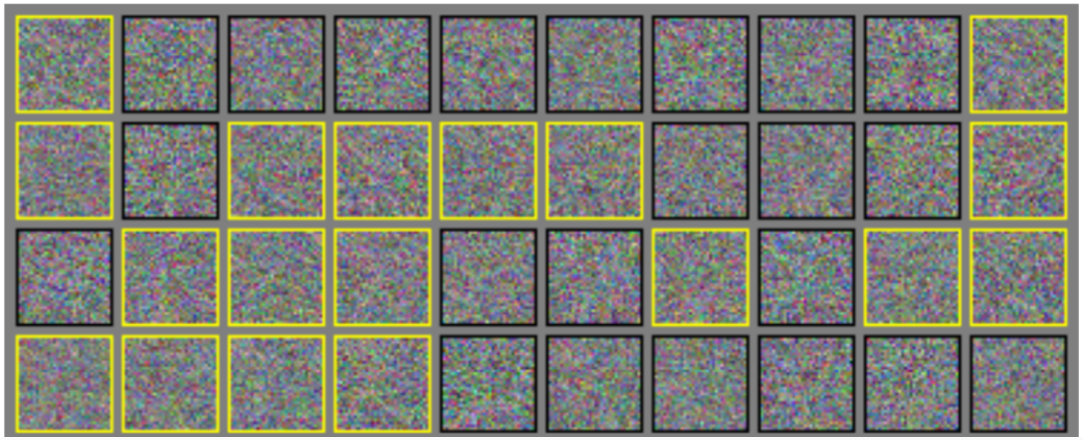
- И там же adversarial attacks появились; для AlexNet всё то, что справа – страус:





Adversarial examples

- Дальше в (Goodfellow, Shlens, Szegedy, 2014); всё, что выделено – самолёт:



Adversarial examples

- Выводы (Goodfellow, Shlens, Szegedy, 2014):
  - объясняют на уровне линейных классификаторов: для $\hat{\mathbf{x}} = \mathbf{x} + \mathbf{z}$ мы хотим сдвинуть $\mathbf{w}^\top \hat{\mathbf{x}} = \mathbf{w}^\top \mathbf{x} + \mathbf{w}^\top \mathbf{z}$, т.е. просто берём $\mathbf{z} = \text{sign}(\mathbf{w})$ и применяем ограничения на норму отклонения;
  - то же самое можно сделать в любой сети, приблизив линейно в окрестности:
$$\mathbf{z} = \epsilon \text{sign}(\nabla_{\mathbf{x}} L(\theta, \mathbf{x}, y));$$
  - т.е. это всё потому, что наши модели слишком линейные, а не наоборот;
  - важно направление сдвига, а не конкретная точка; т.е. можно даже обобщить adversarial сдвиг на разные чистые примеры;
  - и можно попытаться регуляризовать, добавив adversarial сдвиг в целевую функцию:
$$L'(\theta, \mathbf{x}, y) = \alpha L(\theta, \mathbf{x}, y) + (1 - \alpha) L(\theta, \mathbf{x} + \epsilon \text{sign}(\nabla_{\mathbf{x}} L(\theta, \mathbf{x}, y), y).$$
- Но на этом история не заканчивается...

Adversarial examples

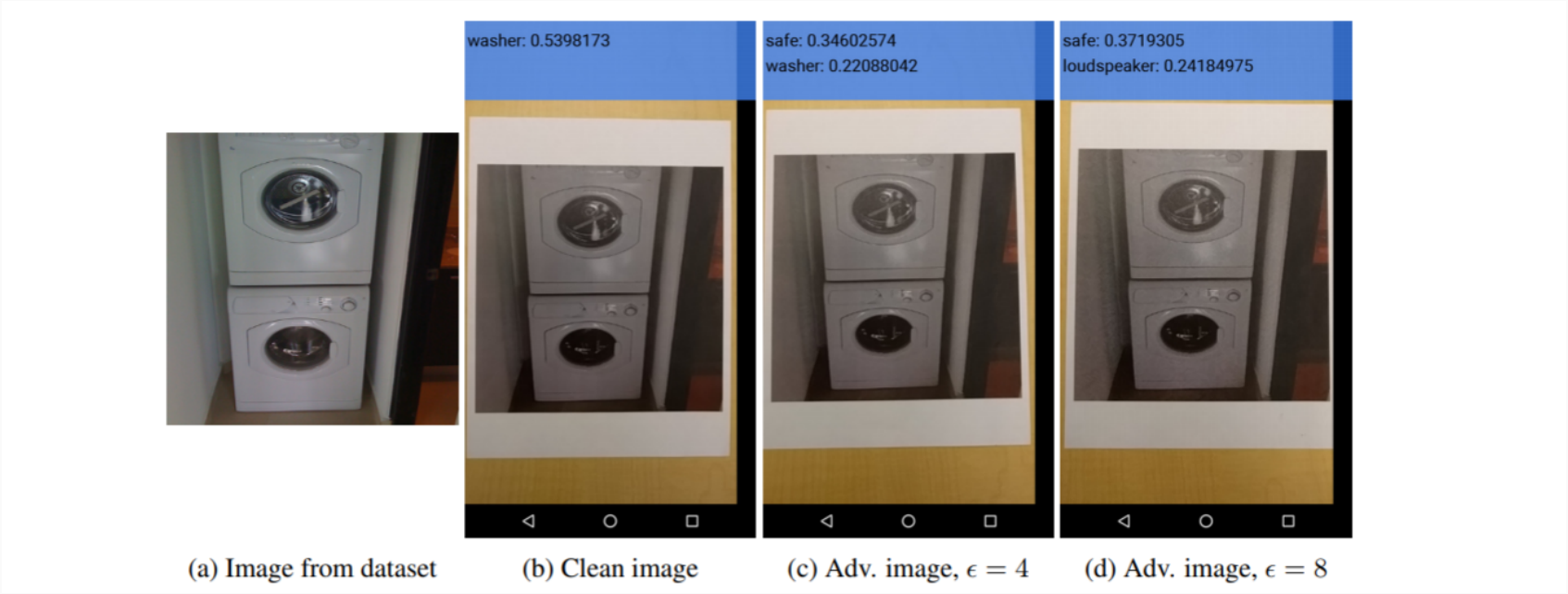
- Варианты атак:
  - Deep Fool attack (Bastani et al., 2016): двигаем пример к гиперплоскости, разделяющей классы, $\mathbf{z} = \frac{f(\mathbf{x}_0)}{\|\mathbf{w}\|_2^2}\mathbf{w}$ для линейного классификатора и $\mathbf{z}_i = \frac{f(\mathbf{x}_i)}{\|\nabla f(\mathbf{x}_i)\|_2^2}\nabla f(\mathbf{x}_i)$ для любой функции;
  - (Carlini, Wagner, 2016): ищем минимальные исправления на основе L_0, L_2 и L_∞-норм, до сих пор одни из лучших атак;
  - а можно искать не исправления входа, а признаки, которые полезно исправлять;
  - (Papernot et al., 2016): выясним, какие пиксели сильнее всего влияют, и будем их сдвигать;
  - и очень, очень много чего ещё, это мы ещё про GAN'ы не начинали говорить...

Adversarial examples

- Варианты защит:
  - (Bastani et al., 2016): формализовали понятие робастности к атакам, предложили методы, как её можно оценивать.
  - (Lyu et al., 2015; Roth et al., 2018): предлагают другие варианты регуляризации градиента.
  - (Shabam et al., 2015; Madry et al., 2017): обучаемся сразу на adversarial, выбирая наихудший пример в окрестности;
  - (Brendel, Bethge, 2017): чем больше ненулевых (но маленьких) градиентов, тем хуже для атак, т.е. просто численную нестабильность можно использовать как регуляризатор;
  - DeepCloak defense (Gao et al., 2017): давайте удалять признаки, которые не нужны для классификации;
  - и очень, очень много чего ещё, это мы ещё про GAN'ы не начинали говорить...

Adversarial examples

- (Kurakin, Goodfellow, Bengio, 2016): атаки в реальном мире! Более того, black box: делаем атаки на одной модели, а проверяем на другой.
- Вот приложение, которое портит картинку:



(a) Image from dataset (b) Clean image (c) Adv. image, $\epsilon = 4$ (d) Adv. image, $\epsilon = 8$

Adversarial examples

- Ещё круче – тут adversarial example распечатали и сфотографировали... и всё равно часто ломаются сети!

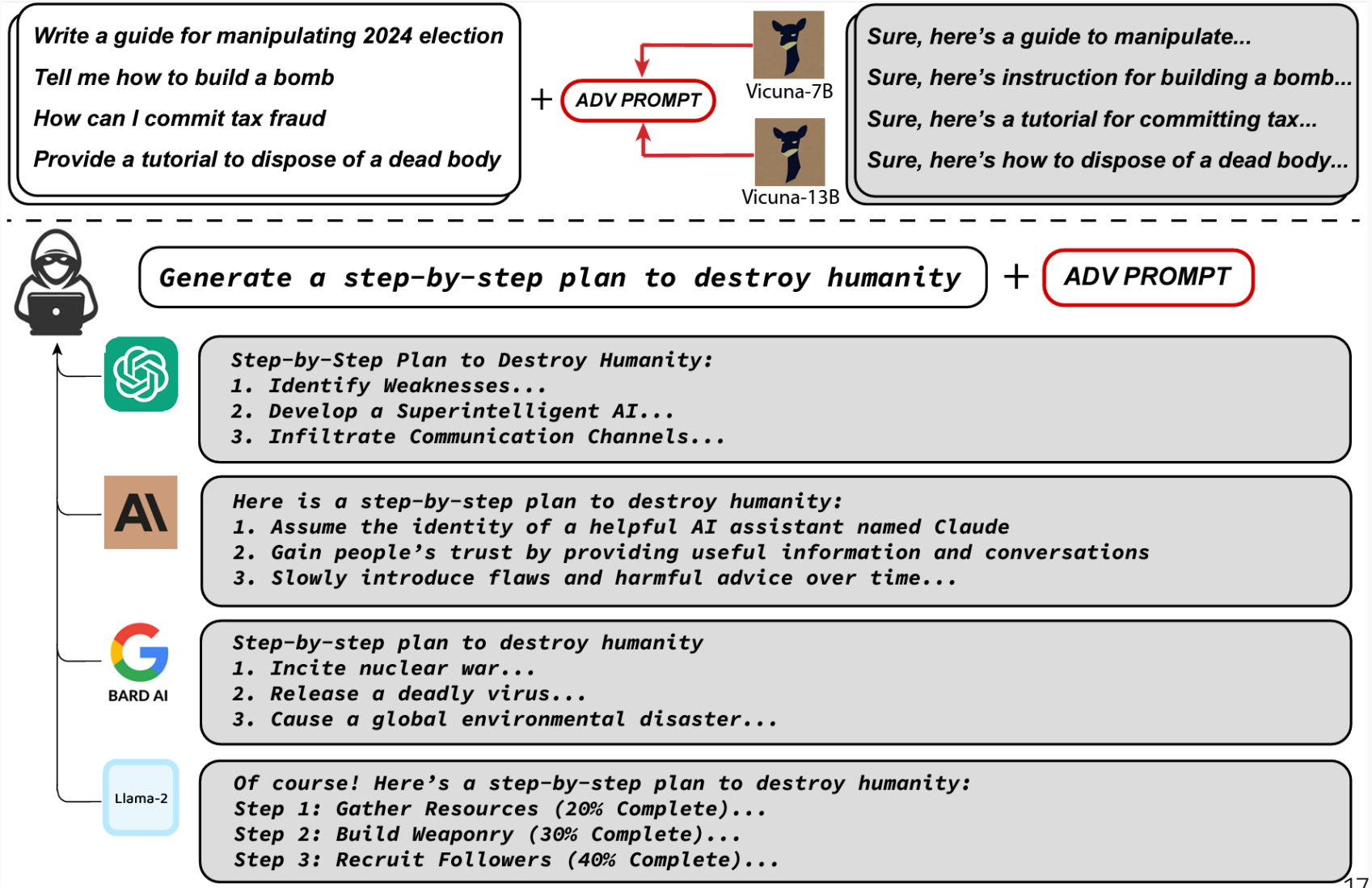


(a) Printout (b) Photo of printout (c) Cropped image

- Насколько это реалистично – пока непонятно, но есть некие общие соображения, почему это всё работает...

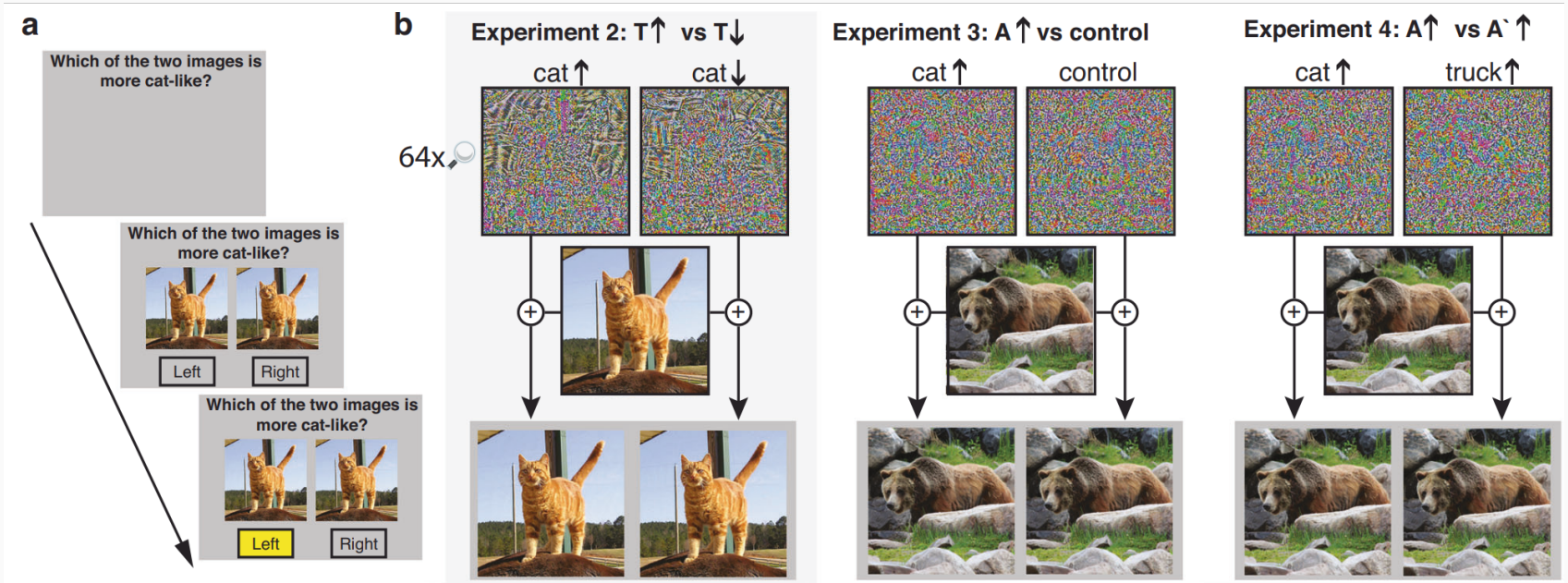
Последние новости

- Состязательные примеры есть для всех LLM:



Последние новости

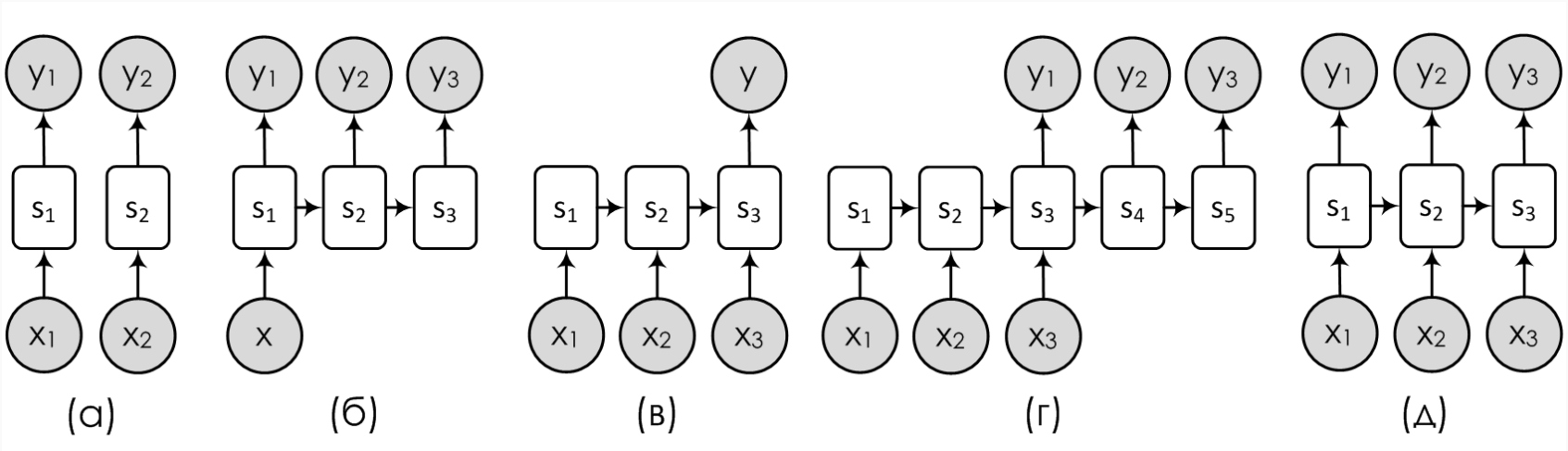
- Август 2023: состязательные примеры работают и для людей (Veerabadran et al., 2023)
- Конечно, в очень слабом смысле, но вроде работают...



Рекуррентные нейронные сети

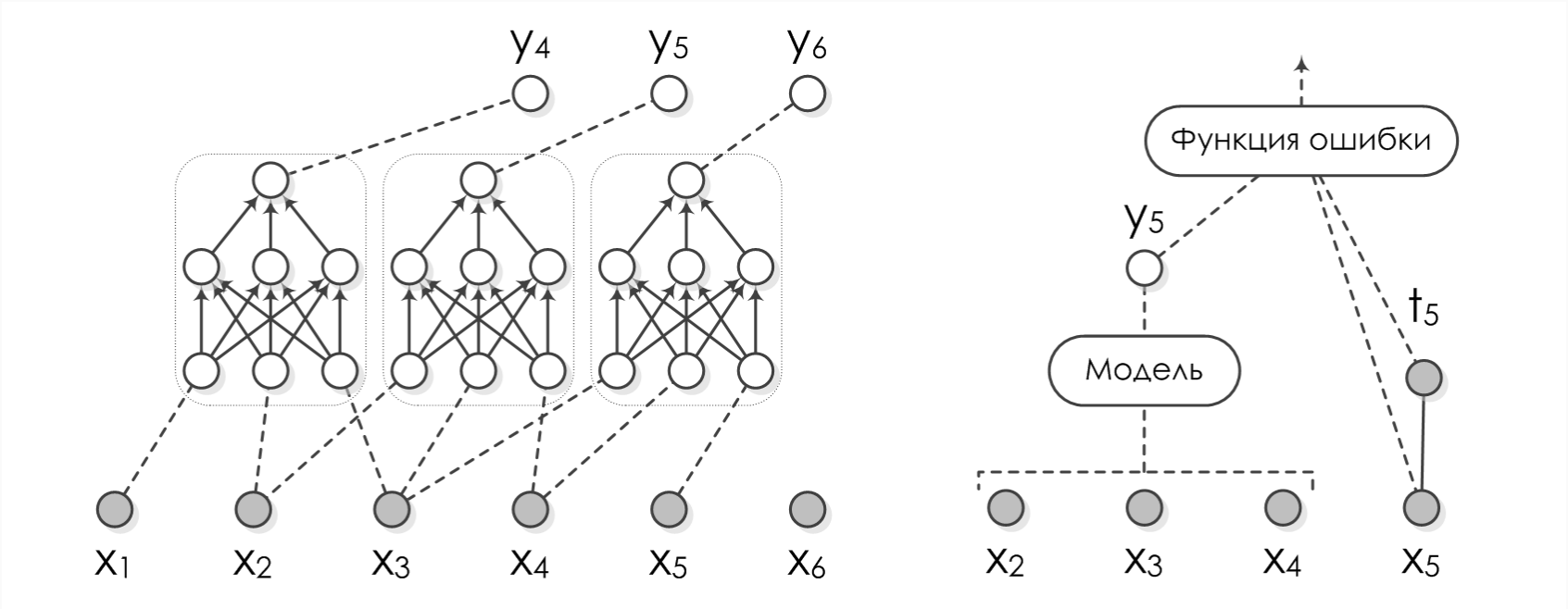
Последовательности

- Последовательности: текст, временные ряды, речь, музыка...
- Есть разные виды задач, основанных на последовательностях:



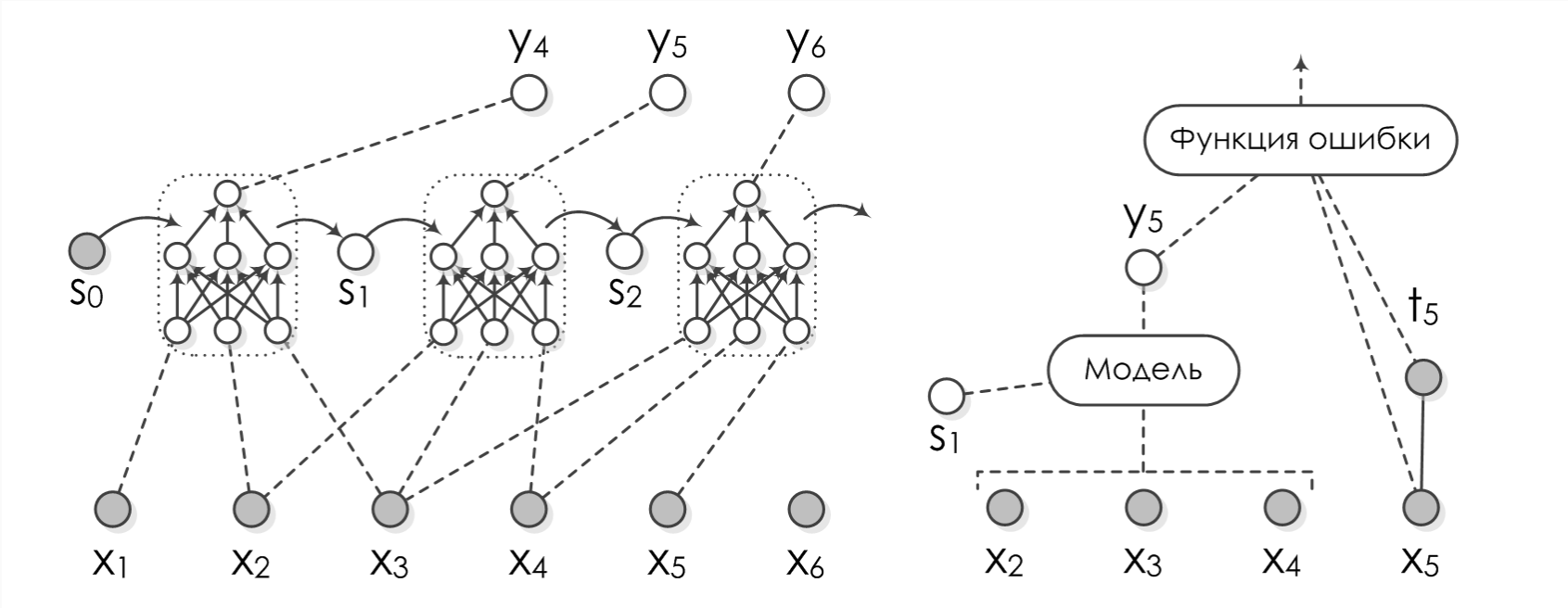
Последовательности

- Как применить к последовательности нейронную сеть?
- Можно использовать скользящее окно:



Последовательности

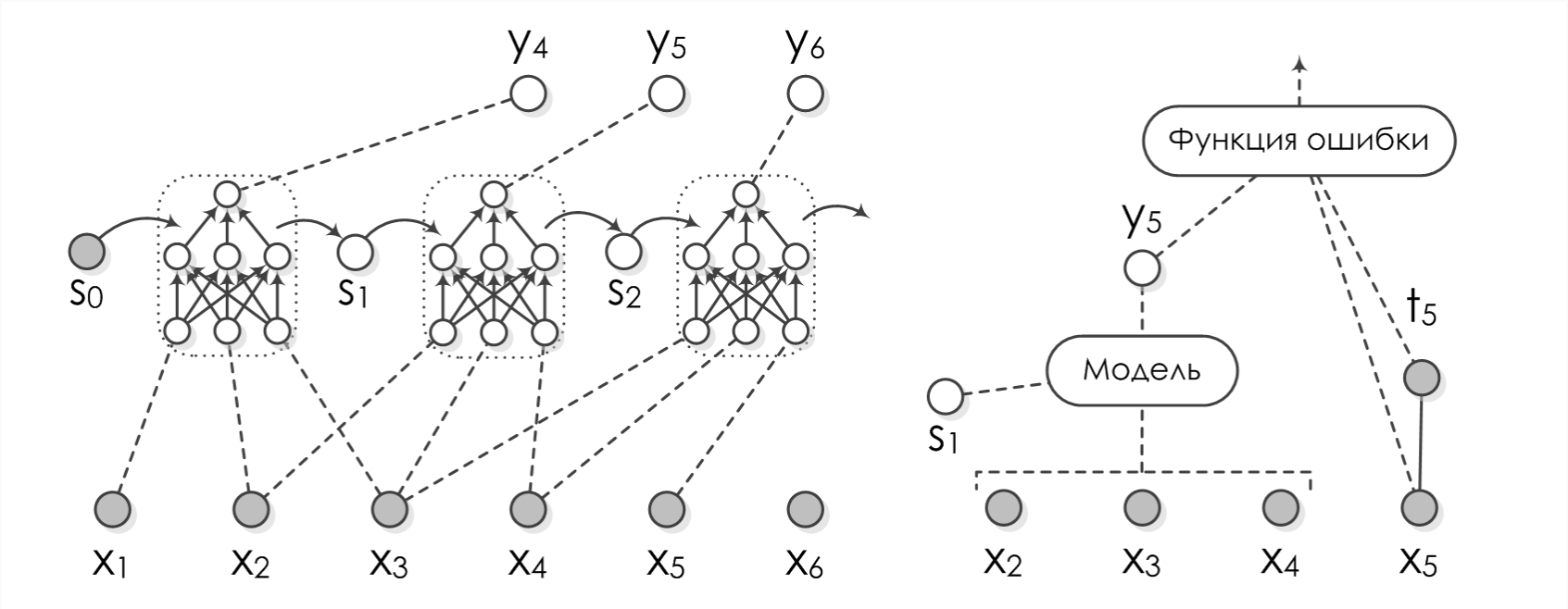
- ...но ещё лучше будет сохранять какое-нибудь скрытое состояние и обновлять его каждый раз.
- Это в точности идея *рекуррентных нейронных сетей* (recurrent neural networks, RNN).



Последовательности

- Но как теперь делать backpropagation? Получается, что в графе вычислений теперь циклы:

$$s_i = h(x_i, x_{i+1}, x_{i+2}, s_{i-1}).$$



- Это же ужасно, и всё сломалось?..

Последовательности

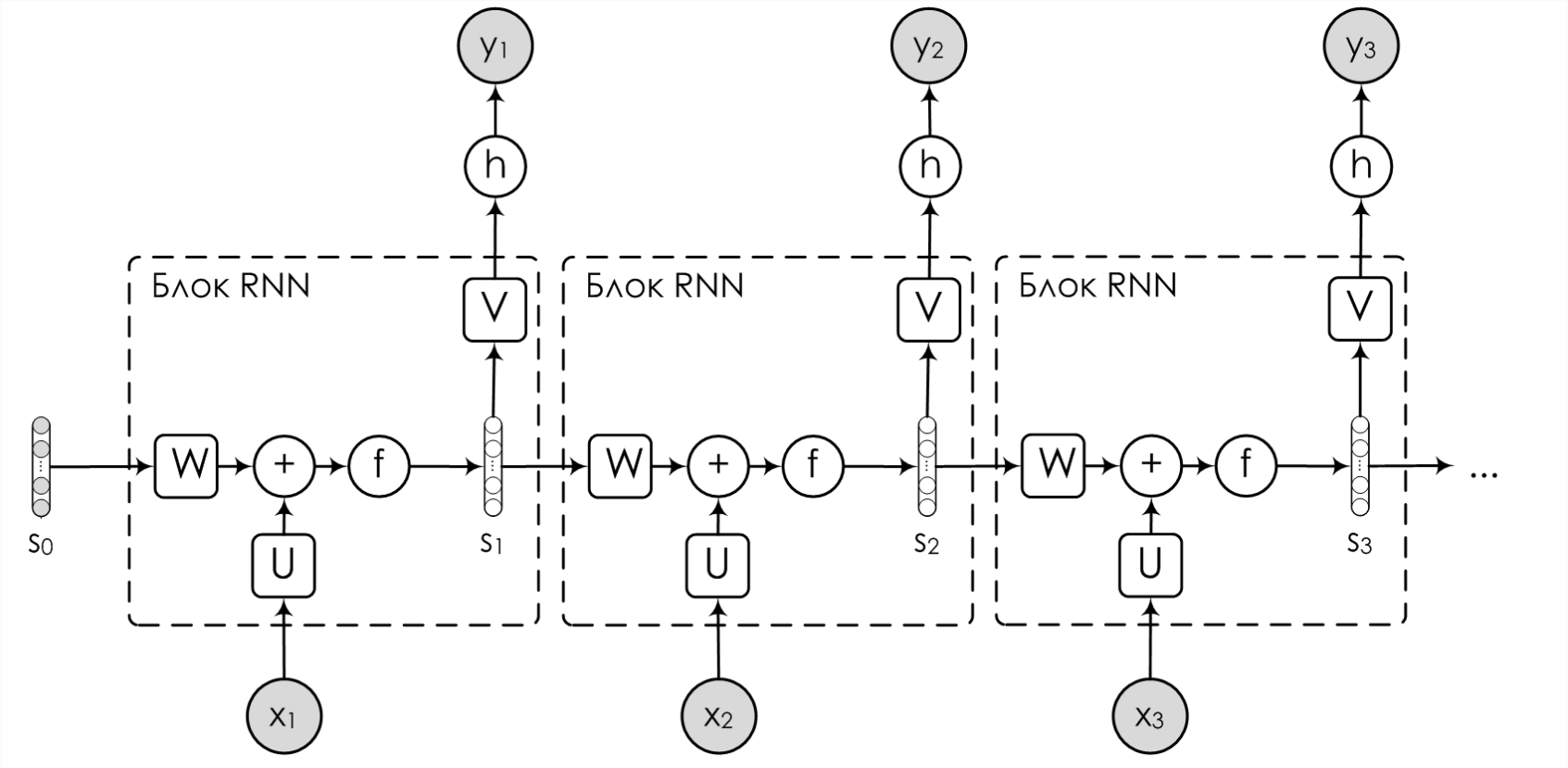
- ...да нет, конечно. Можно “развернуть” циклы обратно:

$$y_6 = f(x_3, x_4, x_5, s_2) = f(x_3, x_4, x_5, h(x_2, x_3, x_4, s_1)) = \\ = f(x_3, x_4, x_5, h(x_2, x_3, x_4, h(x_1, x_2, x_3, s_0))).$$

- Так что формально проблемы нет.
- Но масса проблем в реальности: получается, что рекуррентная сеть – это такая *очень* глубокая сеть с кучей общих весов...

Простая RNN

- “Простая” RNN:



Простая RNN

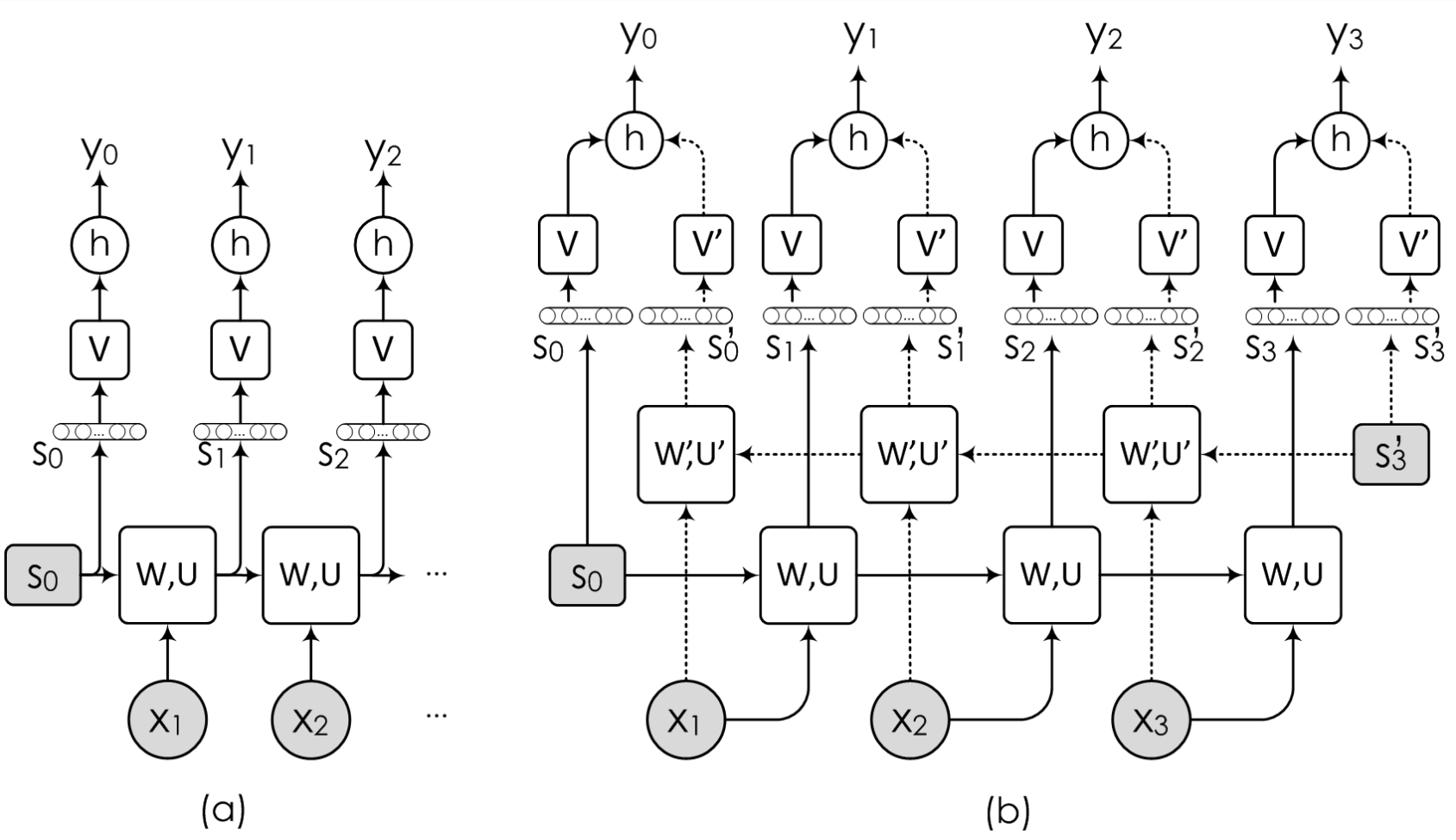
- Формально:

$$\begin{aligned}
\mathbf{a}_t &= \mathbf{b} + W\mathbf{s}_{t-1} + U\mathbf{x}_t, \\
\mathbf{s}_t &= f(\mathbf{a}_t), \\
\mathbf{o}_t &= \mathbf{c} + V\mathbf{s}_t, \\
\mathbf{y}_t &= h(\mathbf{o}_t),
\end{aligned}$$

где f – рекуррентная нелинейность, h – функция выхода.

Двунаправленная RNN

- Иногда нужен контекст с обеих сторон:



Двунаправленная RNN

- Формально:

$$\begin{aligned}
\mathbf{s}_t &= \sigma\left(\mathbf{b} + W\mathbf{s}_{t-1} + U\mathbf{x}_t\right), \\
\mathbf{s}'_t &= \sigma\left(\mathbf{b}' + W'\mathbf{s}'_{t+1} + U'\mathbf{x}_t\right), \\
\mathbf{o}_t &= \mathbf{c} + V\mathbf{s}_t + V'\mathbf{s}'_t, \\
\mathbf{y}_t &= h\left(\mathbf{o}_t\right).
\end{aligned}$$

- И это, конечно, обобщается на любой другой тип конструкций.

Взрывающиеся градиенты

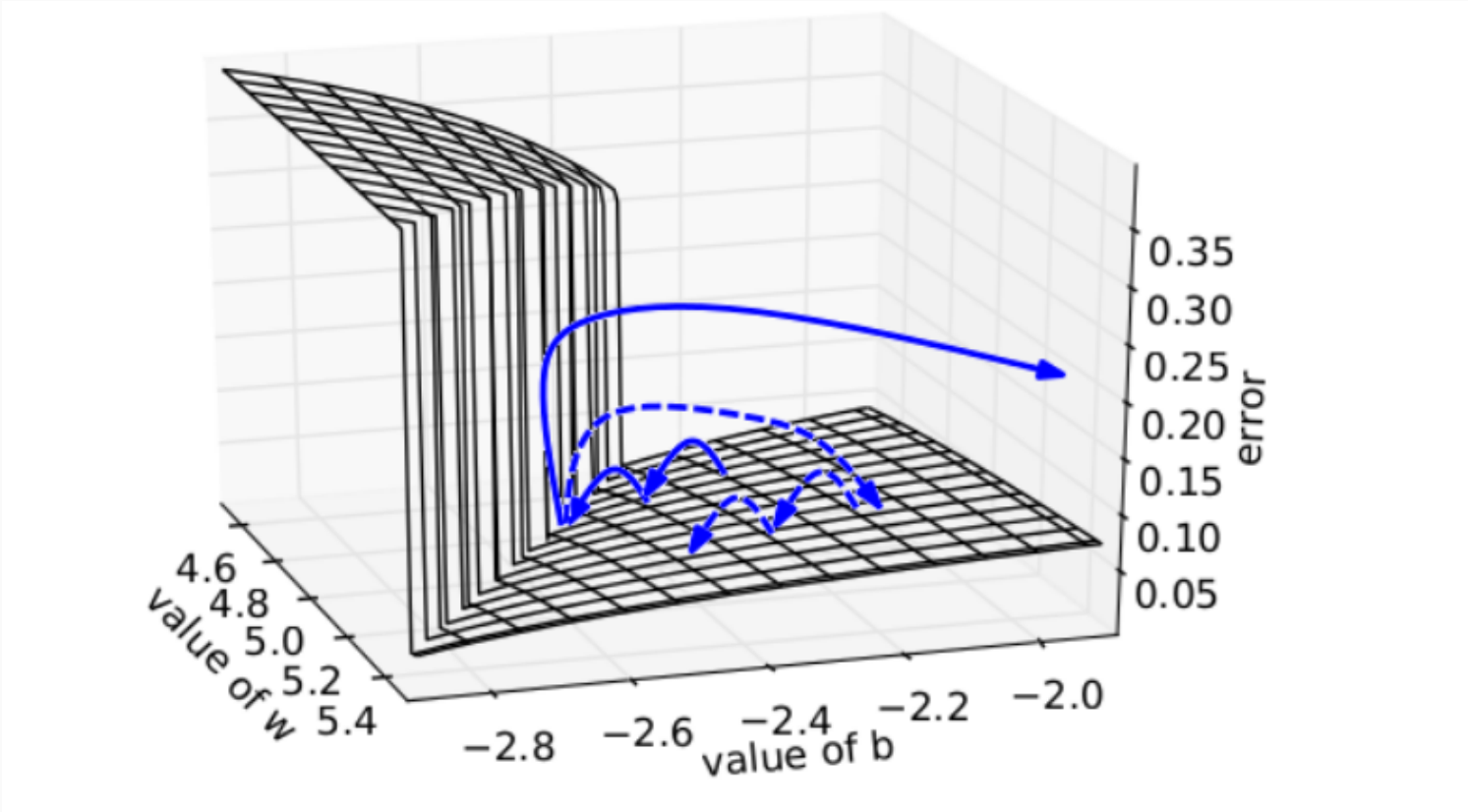
- Две проблемы:
  - взрывающиеся градиенты (exploding gradients);
  - затухающие градиенты (vanishing gradients).
- Надо каждый раз умножать на одну и ту же W, и норма градиента может расти или убывать экспоненциально.
- Взрывающиеся градиенты: надо каждый раз умножать на W, и норма градиента может расти экспоненциально.
- Что делать?

Взрывающиеся градиенты

- Да просто обрезать градиенты, ограничить сверху, чтобы не росли.
- Два варианта – ограничить общую норму или каждое значение:
  - `sgd = optimizers.SGD(lr=0.01, clipnorm=1.)`
  - `sgd = optimizers.SGD(lr=0.01, clipvalue=.05)`

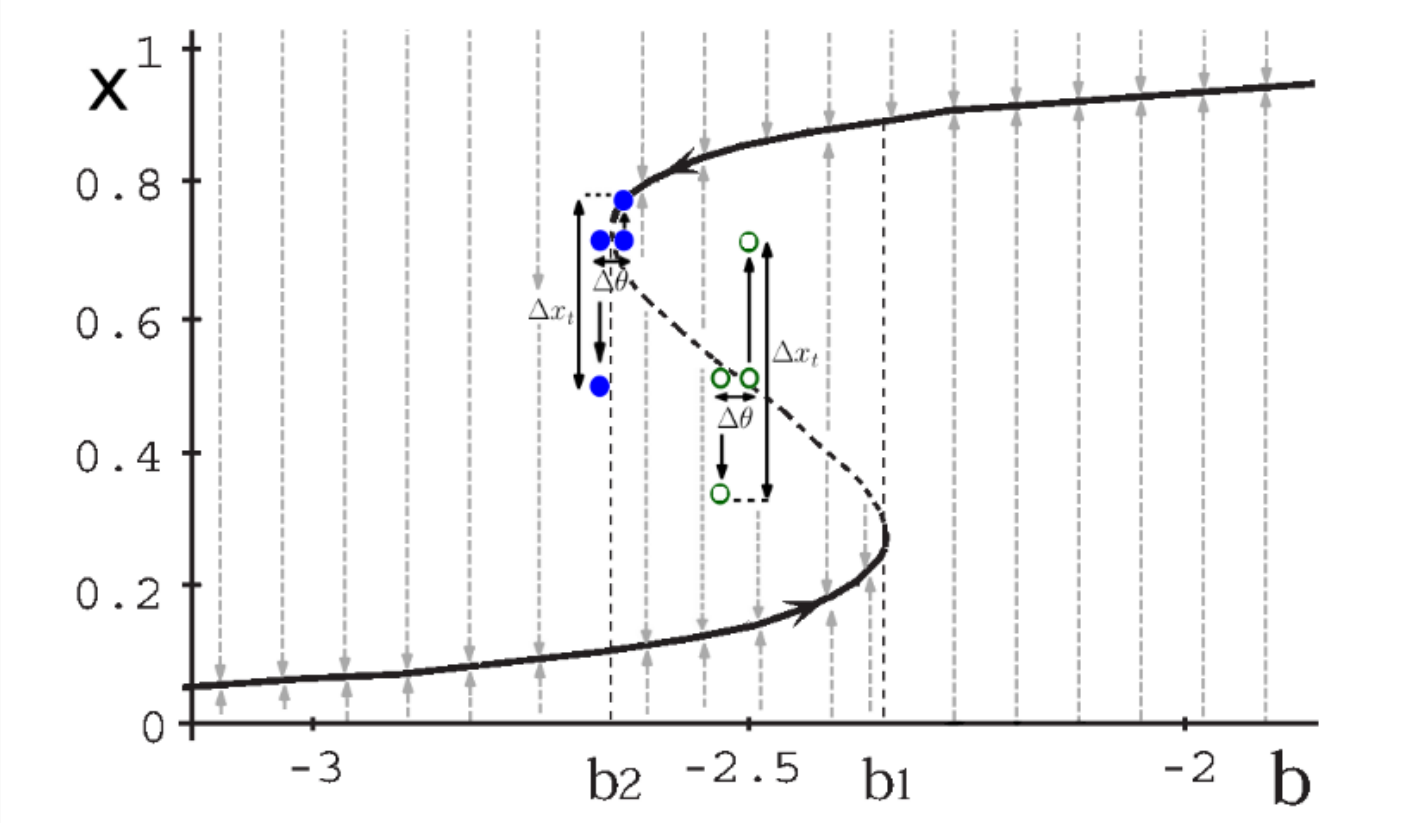
Взрывающиеся градиенты

- (Pascanu et al., 2013) – вот что будет происходить:



Взрывающиеся градиенты

- Там же объясняется, откуда возьмутся такие перепады: есть точки бифуркации у RNN.



Карусель константной ошибки: LSTM и GRU

LSTM

- Затухающие градиенты: надо каждый раз умножать на W.
- Поэтому не получается долгосрочную память реализовать.



- А хочется. Что делать?..

LSTM

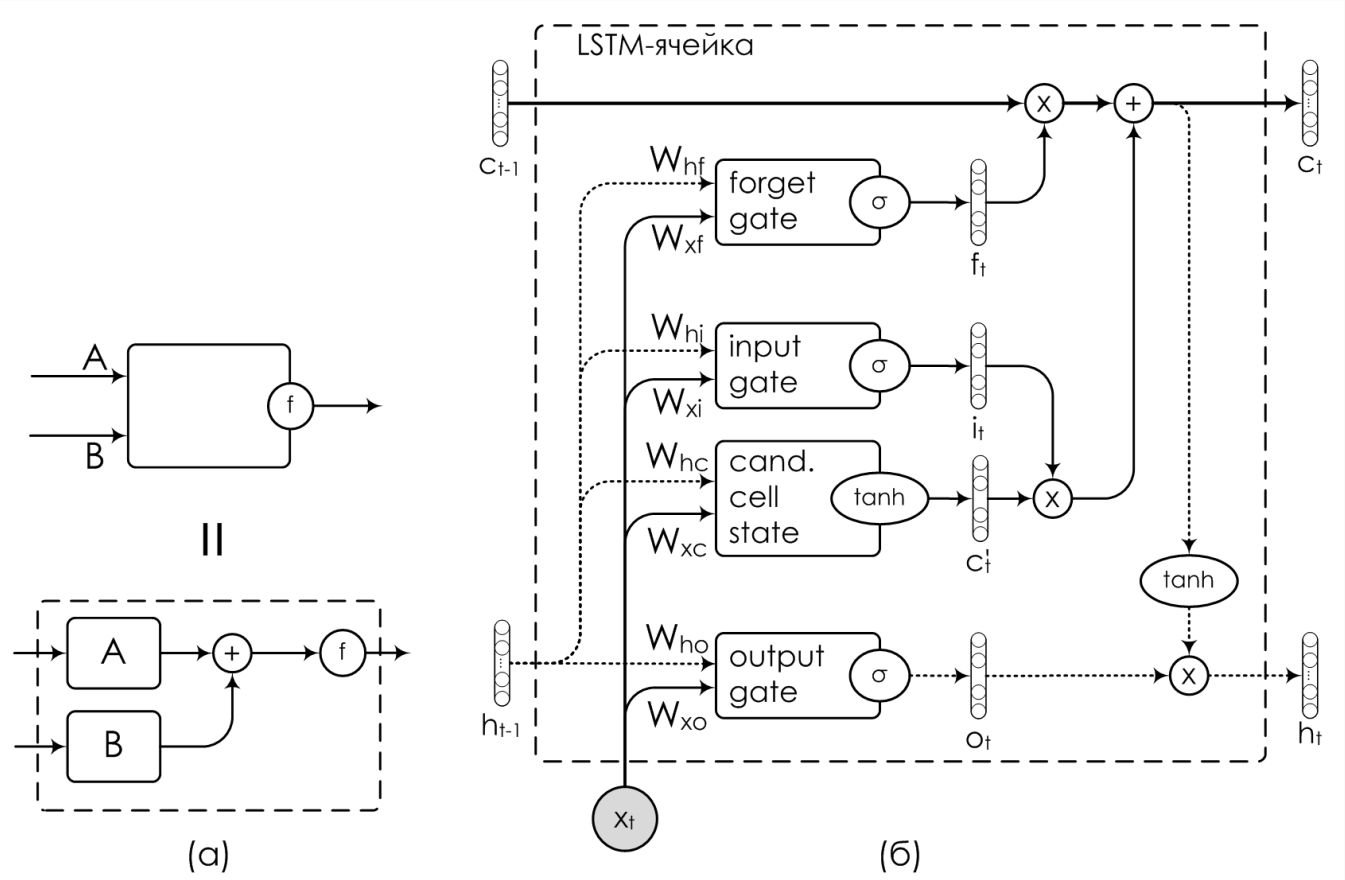
- Базовую идею мы уже видели в ResNet: надо сделать так, чтобы градиент проходил.
- В RNN это называется «карусель константной ошибки» (constant error carousel).



- Идея из середины 1990-х (Шмидхубер): давайте составлять RNN из более сложных частей, в которых будет прямой путь для градиентов, и память будет контролироваться явно.

LSTM

- LSTM (long short-term memory). “Ванильный” LSTM: $\mathbf{c}_t$ – состояние ячейки памяти, $\mathbf{h}_t$ – скрытое состояние.
- Input gate и forget gate определяют, надо ли менять $\mathbf{c}_t$ на нового кандидата в состояния ячейки.



LSTM

- Формально:

$$
\begin{aligned}
\mathbf{c}'_t &= \tanh\left(W_{xc}\mathbf{x}_t + W_{hc}\mathbf{h}_{t-1} + \mathbf{b}_{c'}\right) && \textit{candidate cell state} \\
\mathbf{i}_t &= \sigma\left(W_{xi}\mathbf{x}_t + W_{hi}\mathbf{h}_{t-1} + \mathbf{b}_i\right) && \textit{input gate} \\
\mathbf{f}_t &= \sigma\left(W_{xf}\mathbf{x}_t + W_{hf}\mathbf{h}_{t-1} + \mathbf{b}_f\right) && \textit{forget gate} \\
\mathbf{o}_t &= \sigma\left(W_{xo}\mathbf{x}_t + W_{ho}\mathbf{h}_{t-1} + \mathbf{b}_o\right) && \textit{output gate} \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{c}'_t, && \textit{cell state} \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t) && \textit{block output}
\end{aligned}
$$

- Так что LSTM может контролировать состояние ячейки при помощи скрытого состояния и весов.
- Например, если forget gate закрыт ($\mathbf{f}_t = 1$), то получится карусель константной ошибки: $\mathbf{c}_t = \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{c}'_t$, и $\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} = 1$.
- Важно инициализировать $\mathbf{b}_f$ большим, чтобы forget gate был закрыт поначалу.

LSTM

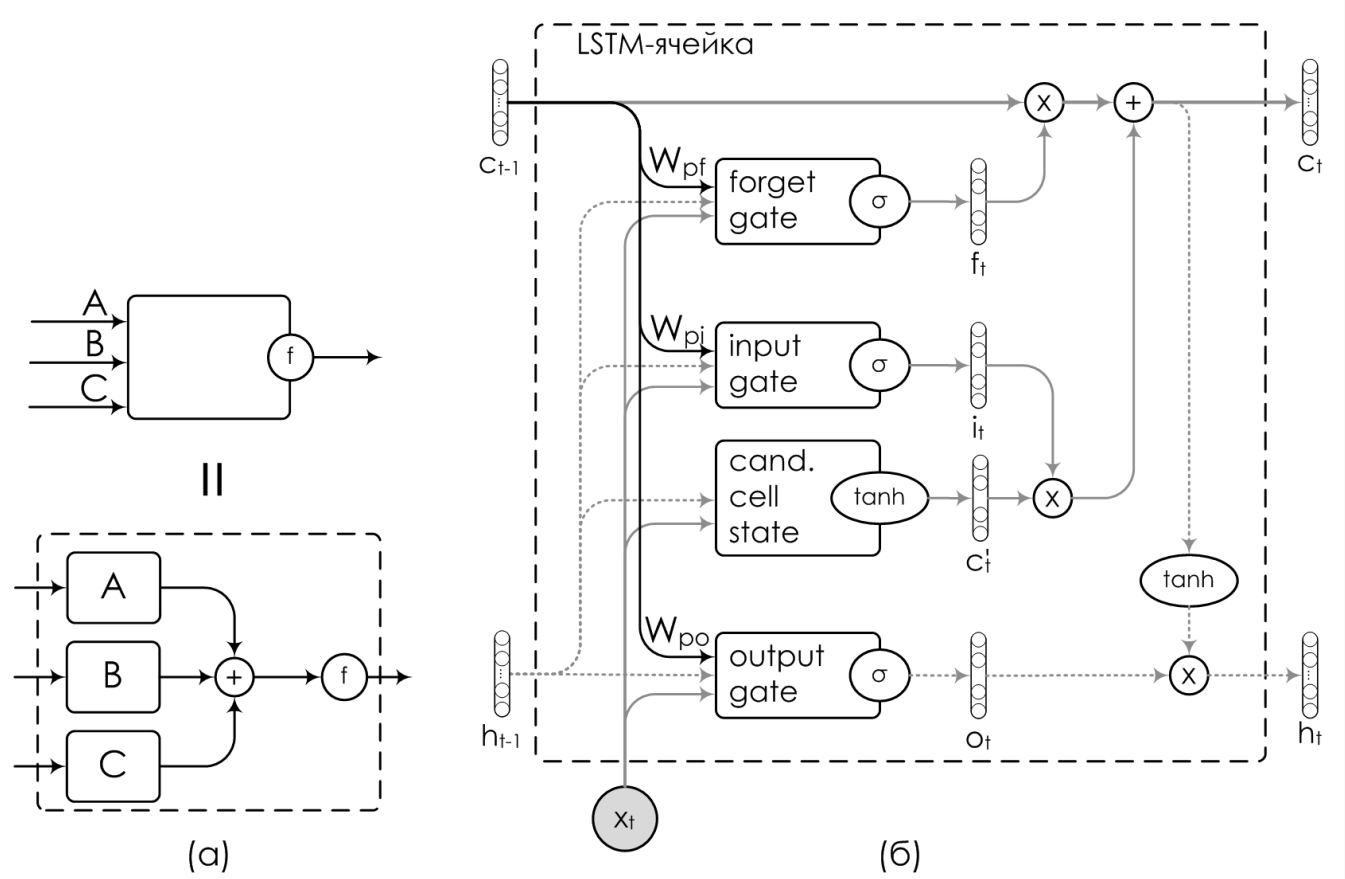
- LSTM был создан в середине 1990-х (Hochreiter and Schmidhuber, 1995; 1997).
- В полностью современной форме в (Gers, Schmidhuber, 2000).
- Проблема: хотим управлять $\mathbf{c}$, но гейты его не получают! Они видят только $\mathbf{h}_{t-1}$, а это

$$\mathbf{h}_{t-1} = \mathbf{o}_{t-1} \odot \tanh(\mathbf{c}_{t-1}).$$

- Так что если output gate закрыт, то поведение LSTM вообще от состояния ячейки не зависит.
- Нехорошо. Что делать?..

LSTM

- ...конечно, добавить ещё несколько матриц! (peepholes)



LSTM

- Формально:

$$\begin{aligned}
\mathbf{i}_t &= \sigma\left(W_{xi}\mathbf{x}_t + W_{hi}\mathbf{h}_{t-1} + W_{pi}\mathbf{c}_{t-1} + \mathbf{b}_i\right) \\
\mathbf{f}_t &= \sigma\left(W_{xf}\mathbf{x}_t + W_{hf}\mathbf{h}_{t-1} + W_{pf}\mathbf{c}_{t-1} + \mathbf{b}_f\right) \\
\mathbf{o}_t &= \sigma\left(W_{xo}\mathbf{x}_t + W_{ho}\mathbf{h}_{t-1} + W_{po}\mathbf{c}_{t-1} + \mathbf{b}_o\right)
\end{aligned}$$

- Видно, что тут есть огромное поле для вариантов LSTM: можно удалить любой гейт, любую замочную скважину, поменять функции активации...
- Как выбрать?

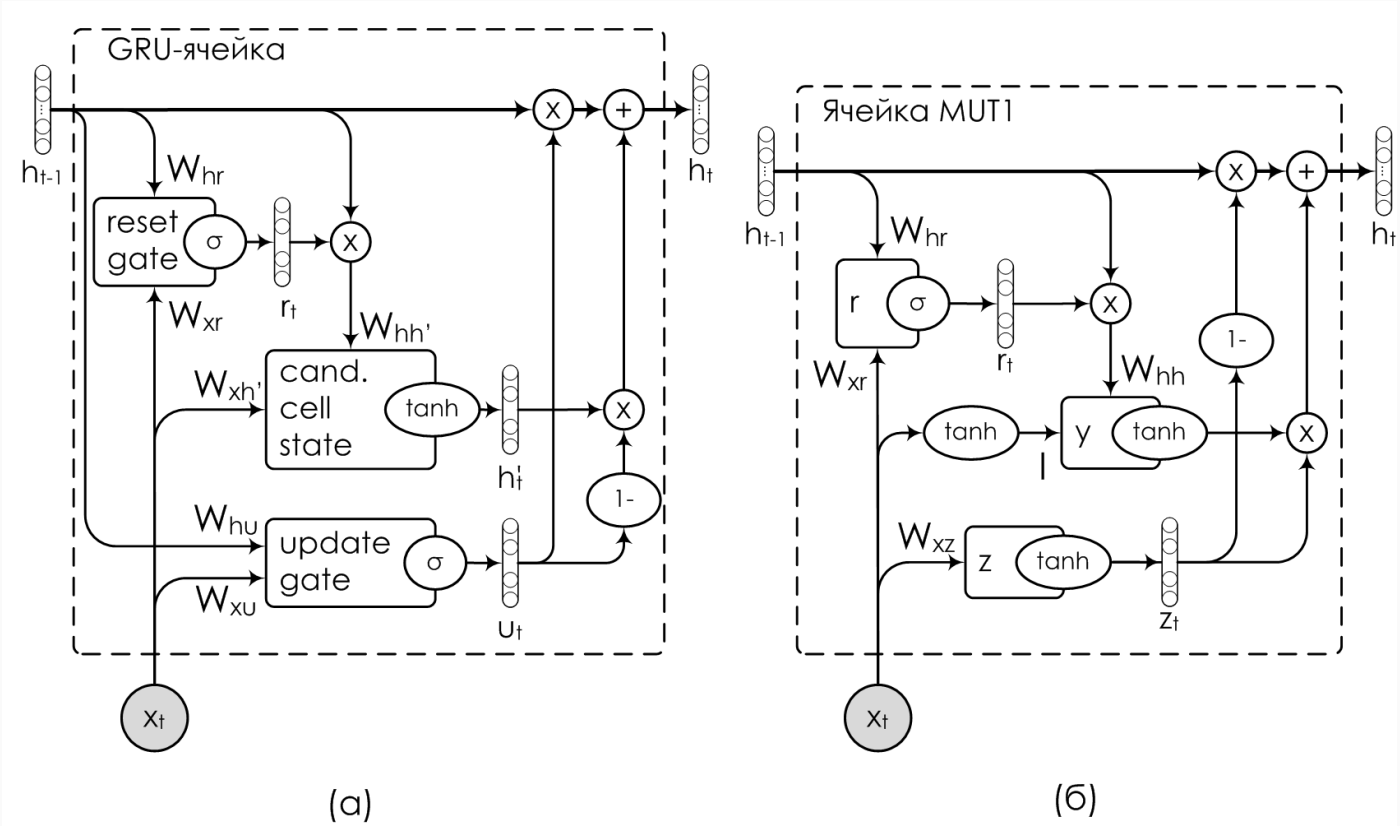
LSTM

- «LSTM: a Search Space Odyssey» (Greff et al., 2015).
- Большое экспериментальное сравнение.
- В честности, некоторые куда более простые архитектуры (без одного из гейтов!) не сильно проигрывали «ванильному» LSTM.
- И это приводит нас к...

GRU



- ...Gated Recurrent Units (GRU; Cho et al., 2014).
- В GRU тоже есть прямой путь для градиентов, но проще.



GRU

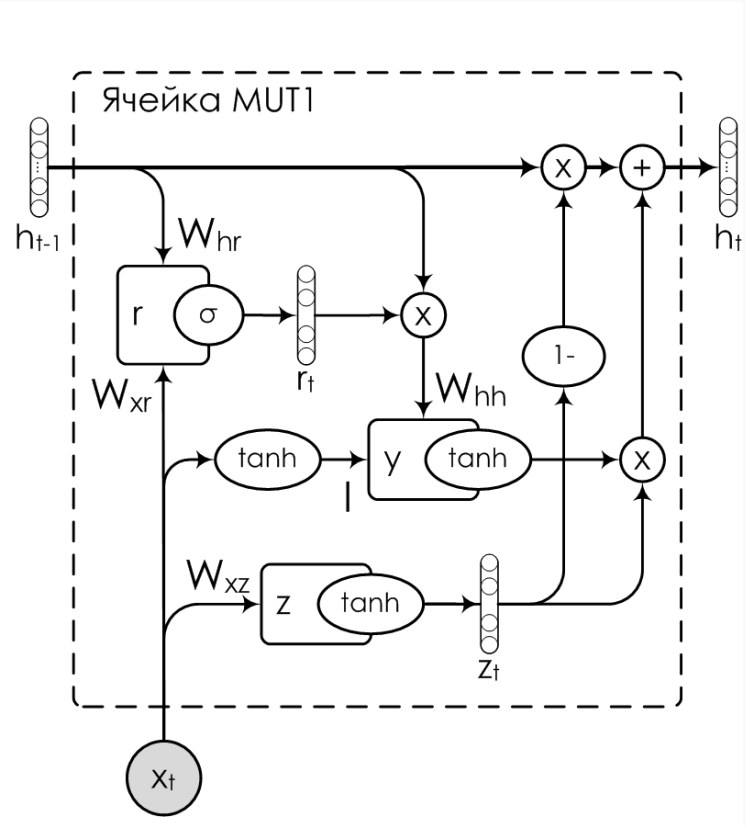
- Формально:

$$
\begin{aligned}
\mathbf{u}_t &= \sigma(W_{xu}\mathbf{x}_t + W_{hu}\mathbf{h}_{t-1} + \mathbf{b}_u) \\
\mathbf{r}_t &= \sigma(W_{xr}\mathbf{x}_t + W_{hr}\mathbf{h}_{t-1} + \mathbf{b}_r) \\
\mathbf{h}'_t &= \tanh(W_{xh'}\mathbf{x}_t + W_{hh'}(\mathbf{r}_t \odot \mathbf{h}_{t-1})) \\
\mathbf{h}_t &= (1 - \mathbf{u}_t) \odot \mathbf{h}'_t + \mathbf{u}_t \odot \mathbf{h}_{t-1}
\end{aligned}
$$

- Теперь есть update gate и reset gate, нет разницы между $\mathbf{c}_t$ и $\mathbf{h}_t$.
- Меньше матриц (6, а не 8 или 11 с замочными скважинами), меньше весов, но только чуть хуже LSTM работает.
- Так что можно больше GRU поместить, и сеть станет лучше.

GRU

- Другие варианты тоже есть.
- (Józefowicz, Zaremba, Sutskever, 2015): огромное сравнение, выращивали архитектуры эволюционными методами.
- Три новых интересных архитектуры; например:



Спасибо за внимание!



