

Задачи компьютерного зрения

Данг Куинь Ны
Станкевич Матвей

17.11.2020

Что будет рассмотрено

- задачи компьютерного зрения
- рассмотрим решения задач с помощью CNN для:
 - детекции (R-CNN, Fast R-CNN, Faster R-CNN, YOLO)
 - сегментации (Mask R-CNN)
 - переноса стиля (Neural Style Transfer)
 - super resolution (SRCNN, SRDenseNet, MS-LapSRN)

Компьютерное зрение

Компьютерное зрение — (Computer Vision, CV) теория и технология создания моделей, которые могут производить обнаружение, отслеживание и классификацию объектов.

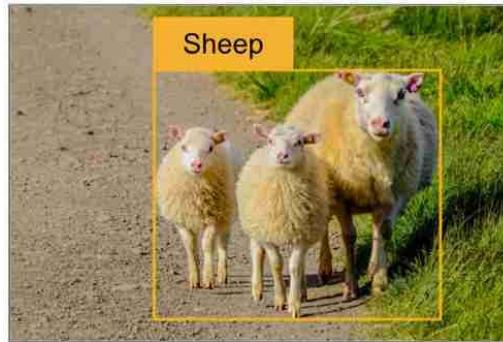
Сфера применения:

- Распознавание лиц
- Беспилотные автомобили
- Дополненная реальность
- Промышленность
- Медицина
- Биометрические данные
- Кино (motion capture)

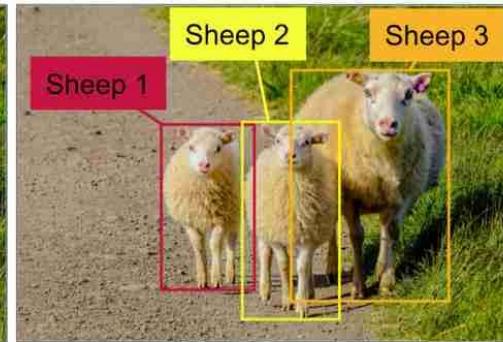
Задачи СV

- Распознавание:
 - Классификация
 - Детекция
 - Оценка положения
 - Оптическое распознавание символов
 - Поиск изображений по содержанию
- Сегментация
- Перенос стиля
- Движение:
 - Визуальная одометрия
 - Трекинг движущегося объекта
- Восстановление и моделирование изображений

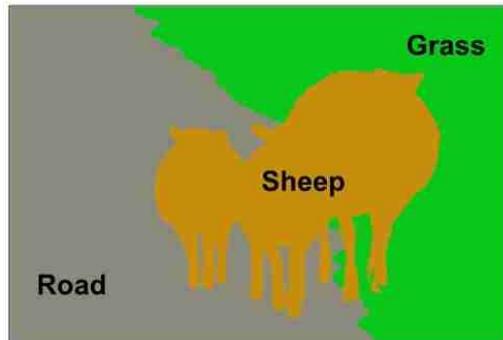
Задачи СВ



Classification + Localization



Object Detection

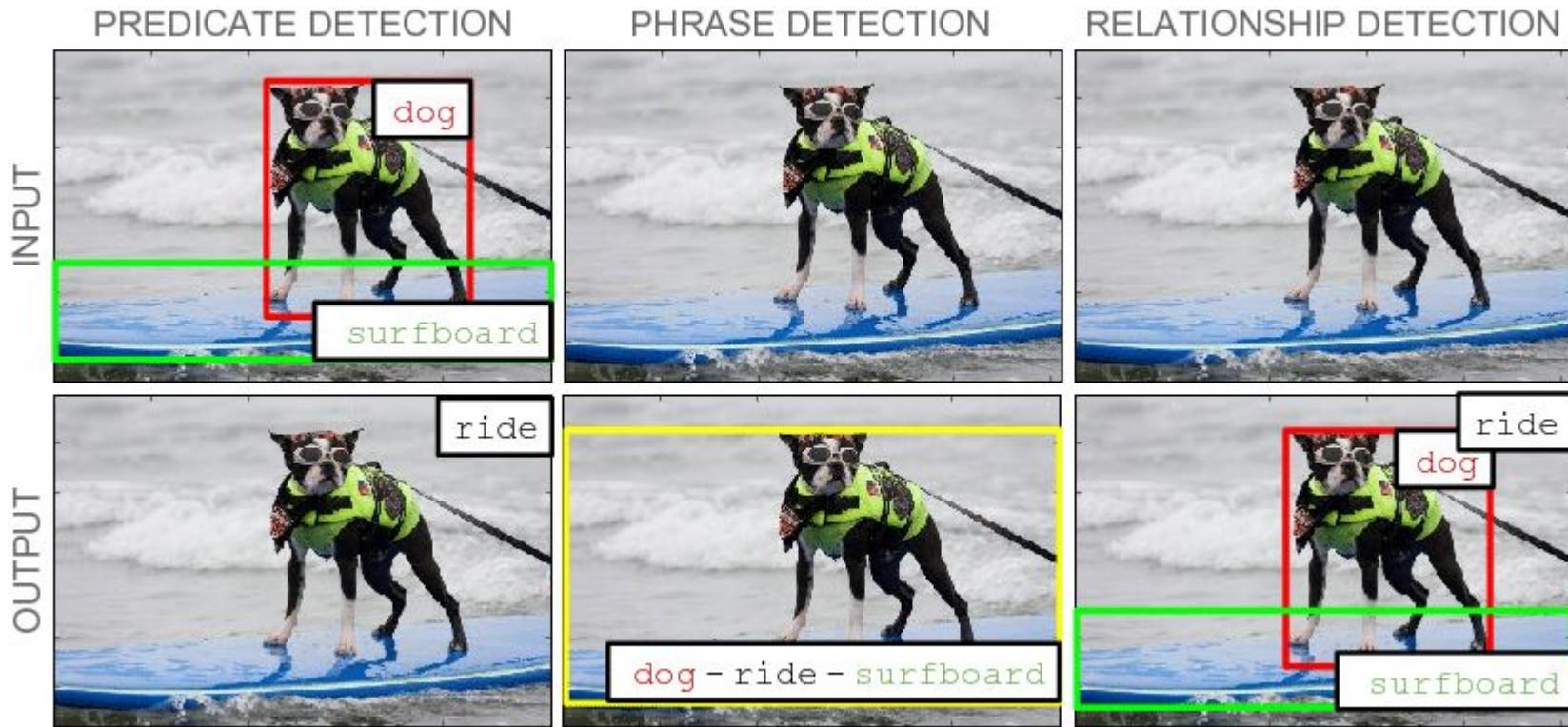


Semantic Segmentation



Instance Segmentation

Задачи CV



Задачи СV

Content image



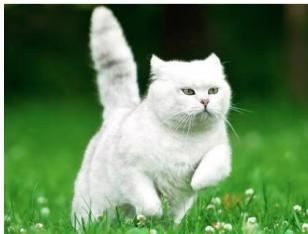
+

Style image



→

Output image



+



→



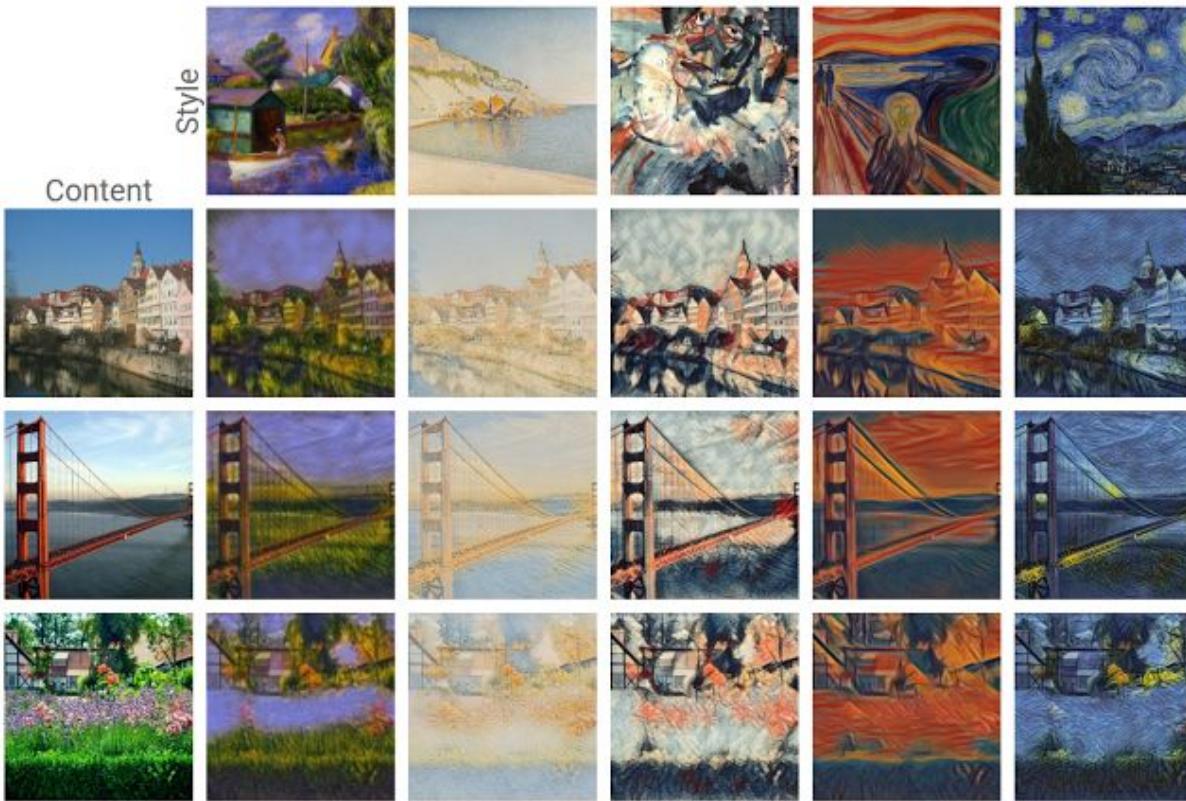
+



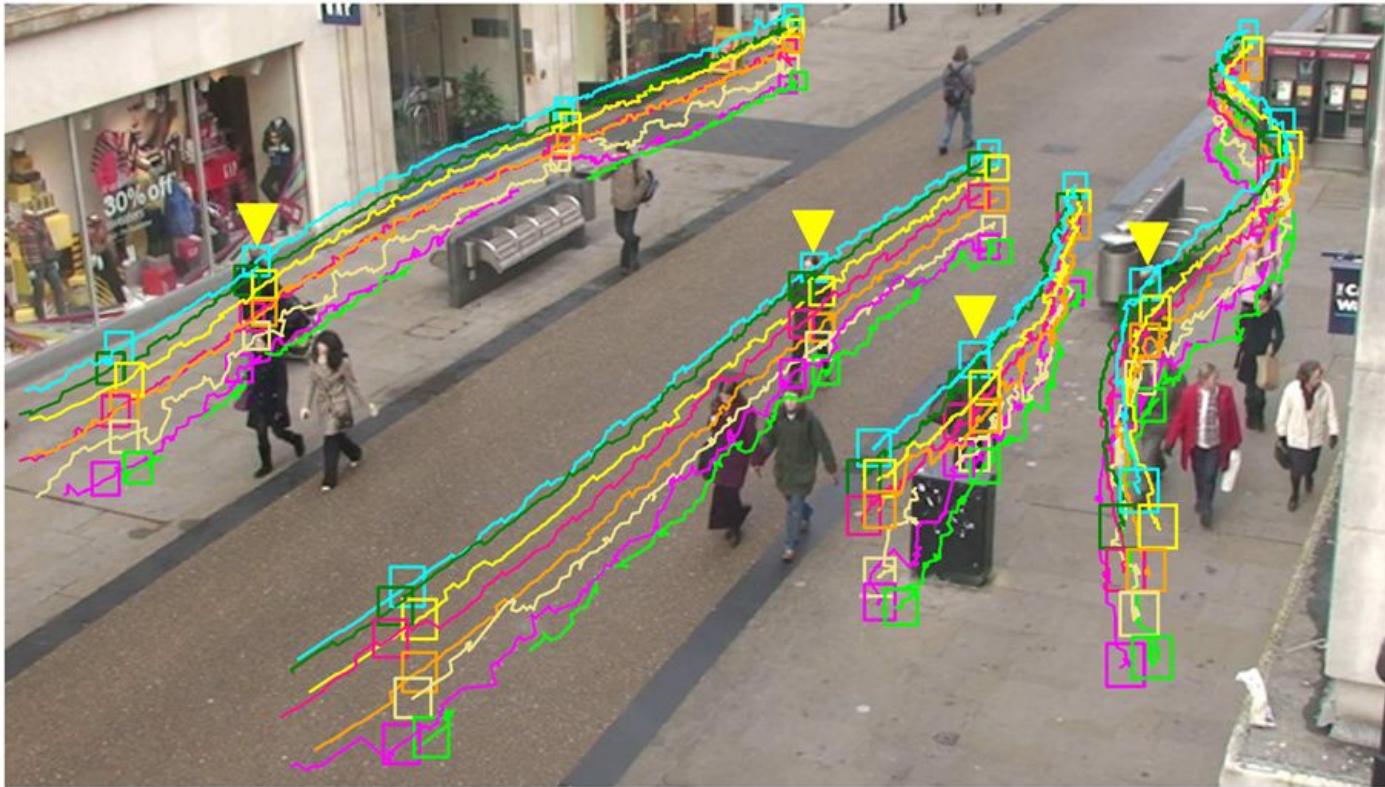
→



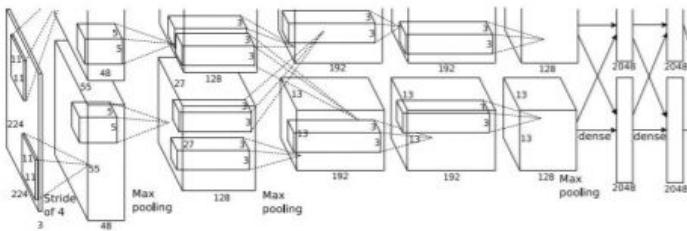
Задачи СV



Задачи СВ



Классификация

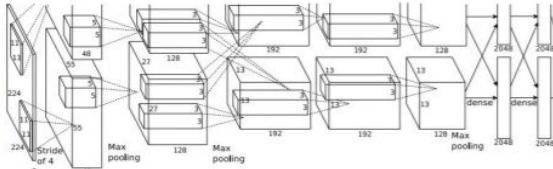


Vector:
4096

Fully-Connected:
4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Классификация + локализация



Fully
Connected:
4096 to 1000

Vector:
4096
Fully
Connected:
4096 to 4

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

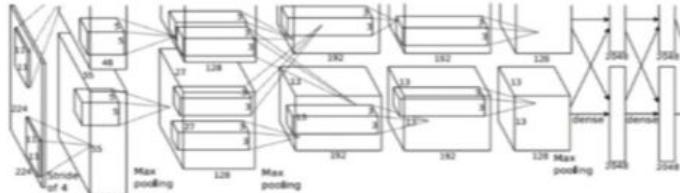
Box
Coordinates → L2 Loss
(x, y, w, h)

Correct label:
Cat

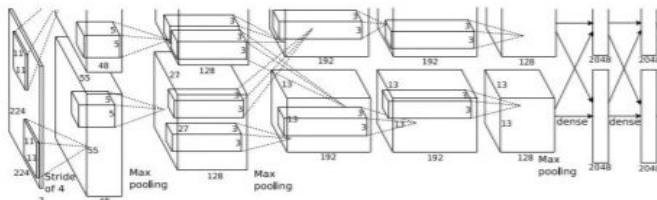
Softmax
Loss

Correct box:
(x', y', w', h')

Детекция



CAT: (x, y, w, h)



DOG: (x, y, w, h)

DOG: (x, y, w, h)

CAT: (x, y, w, h)

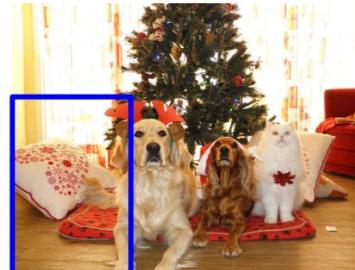
В отличие от локализации, в задаче детекции для каждого объекта разное количество выходов.

Детекция. Скользящие окна

Для каждого окна решаем задачу классификации.

Проблема:

- затратно применять CNN к большому количеству окон (с разными размерами)
- неясно как выбирать окна



Dog? NO
Cat? NO
Background? YES



Dog? YES
Cat? NO
Background? NO

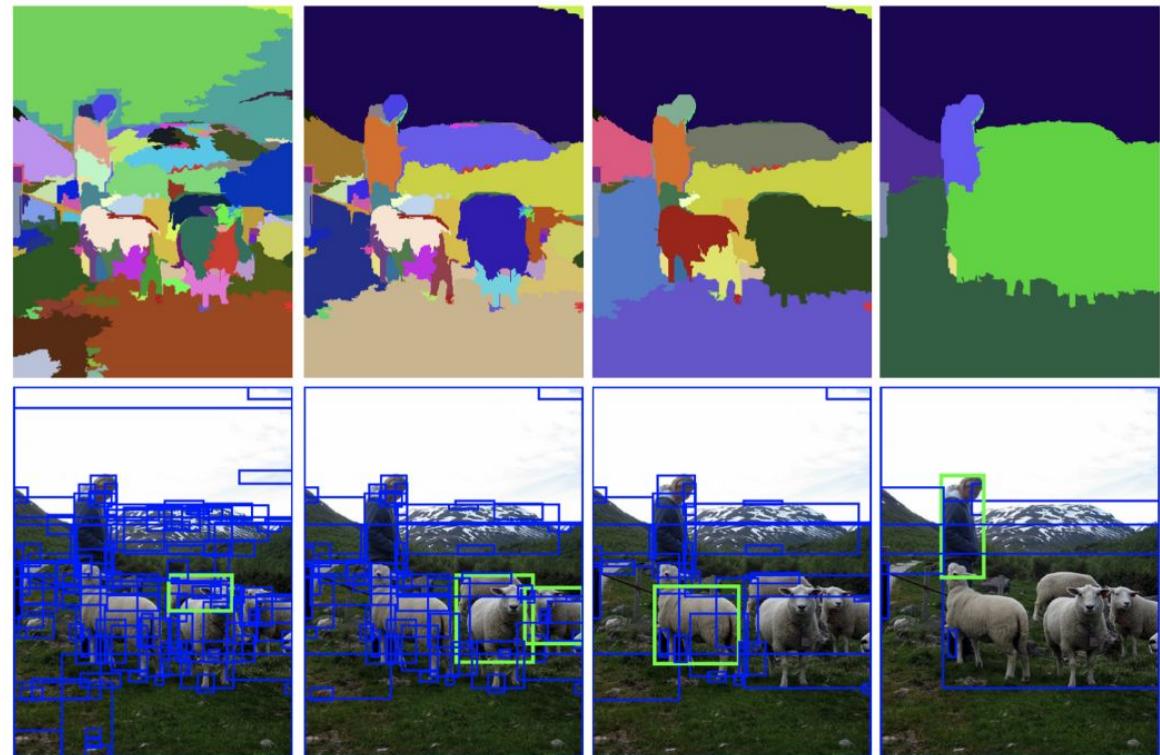


Dog? NO
Cat? YES
Background? NO

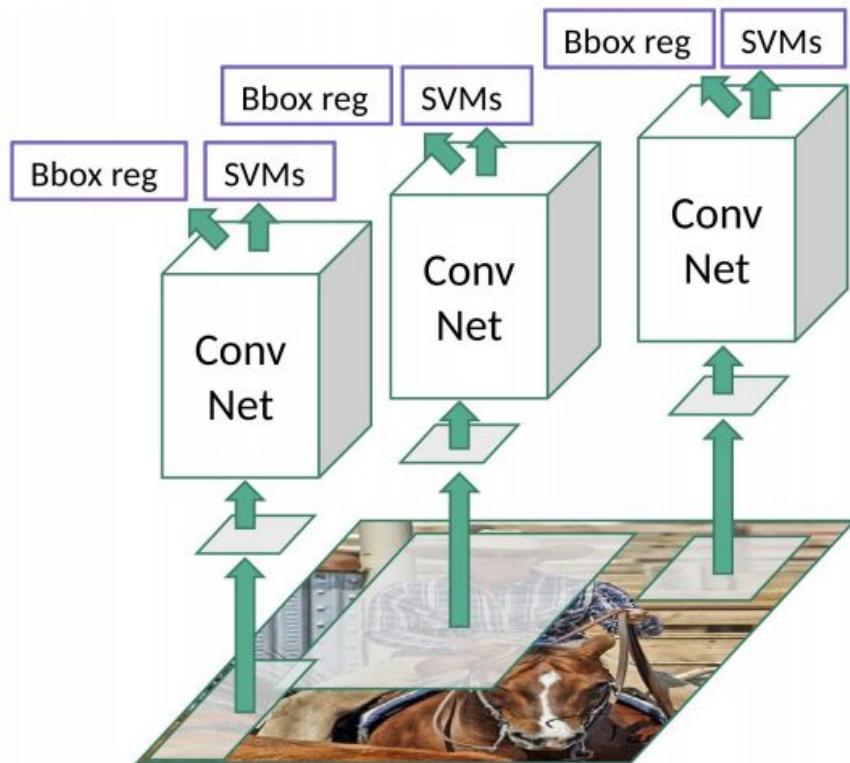
Детекция. Region-based модели

Region proposals:
находим области, где
может быть объект
(RoI - Region of
Interest).

Один из алгоритмов:
Selective Search



Детекция. R-CNN



R-CNN:

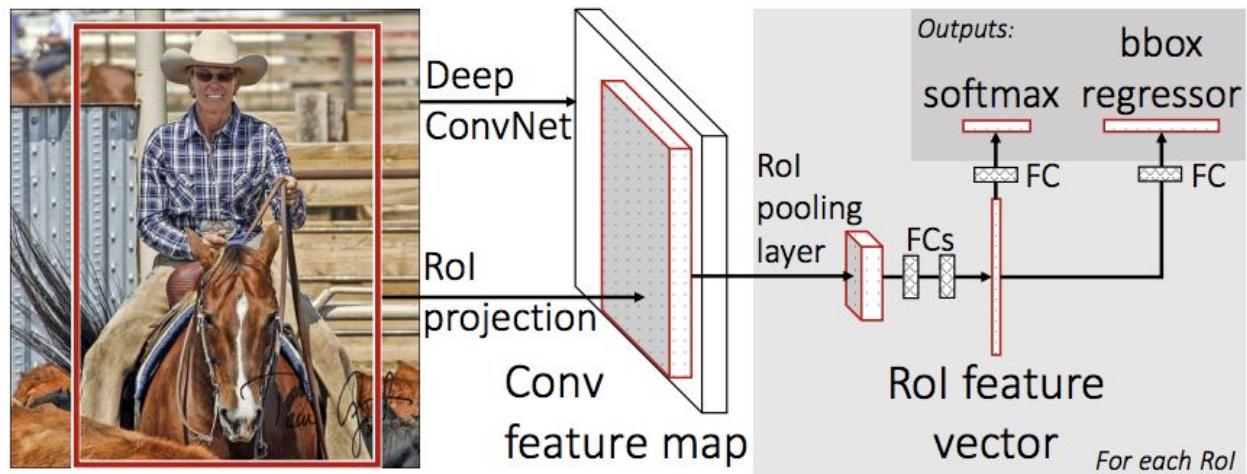
Классификация каждой области + регрессия сдвигов границ

Недостатки:

- Классификация каждой из RoI для каждого изображения затратно
- Выделение RoI без обучения

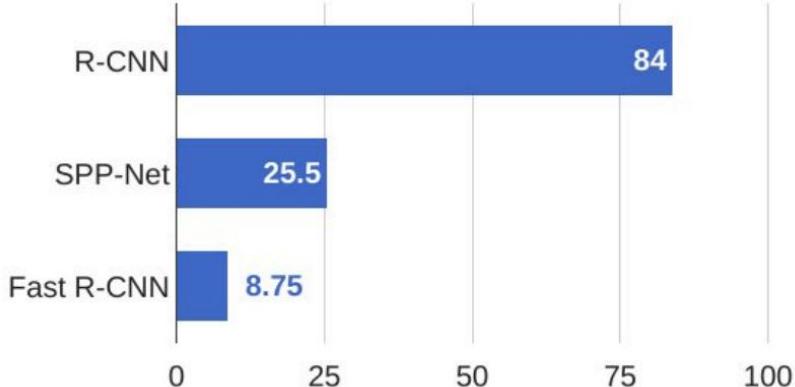
Детекция. Fast R-CNN

Сначала
используем CNN
для всего
изображения,
потом выделяем
RoI.

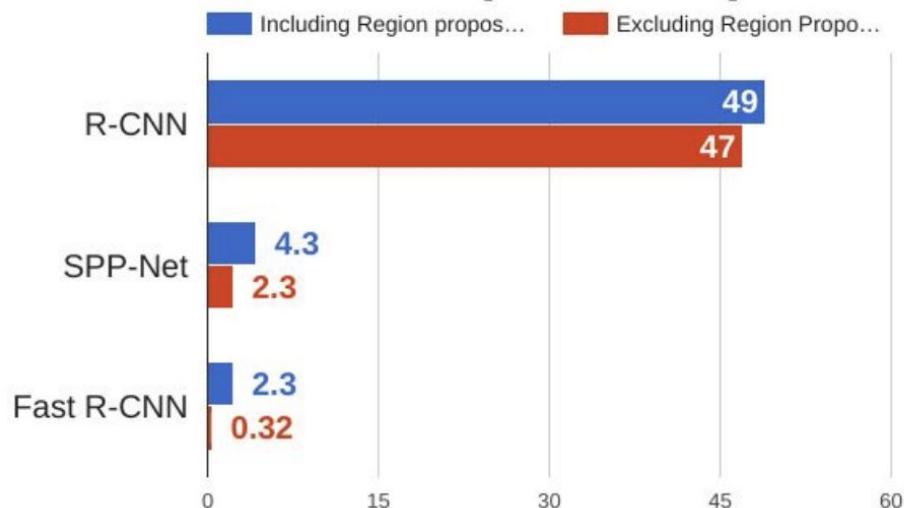


Детекция. R-CNN и Fast R-CNN

Training time (Hours)

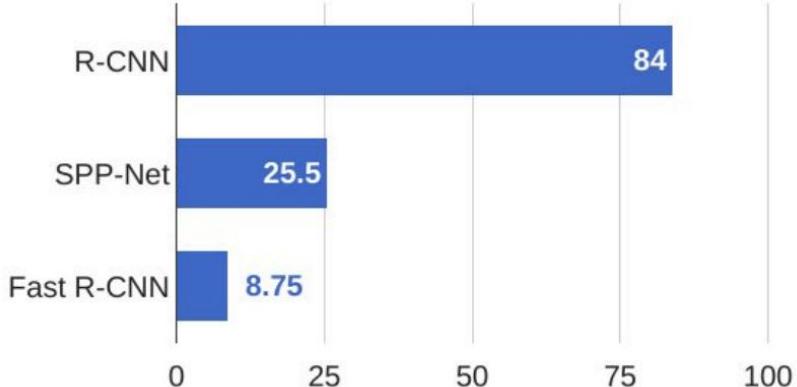


Test time (seconds)

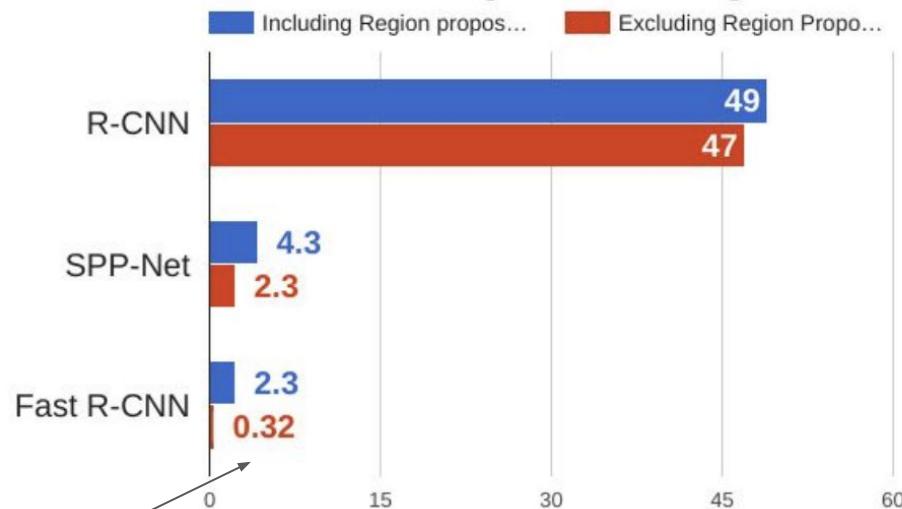


Детекция. R-CNN и Fast R-CNN

Training time (Hours)



Test time (seconds)



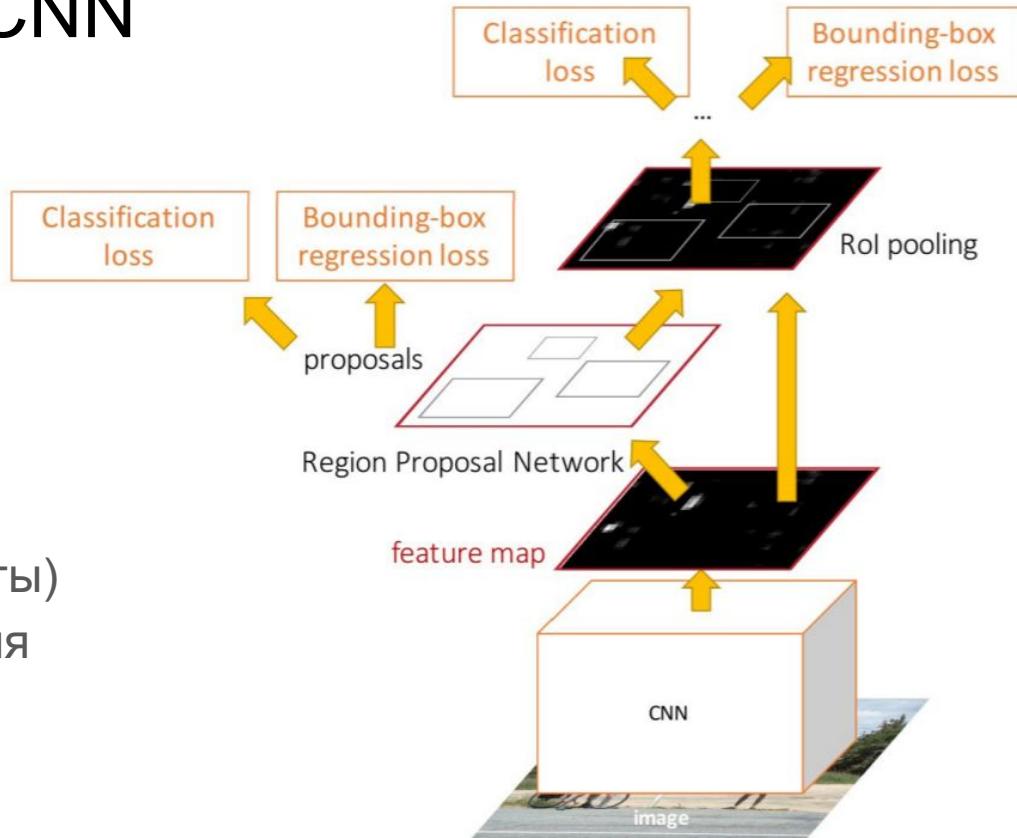
Недостаток: test time существенно зависит от времени выделения ROI

Детекция. Faster R-CNN

Найдём ROI с помощью Region Proposal Network (RPN).

Функции потери:

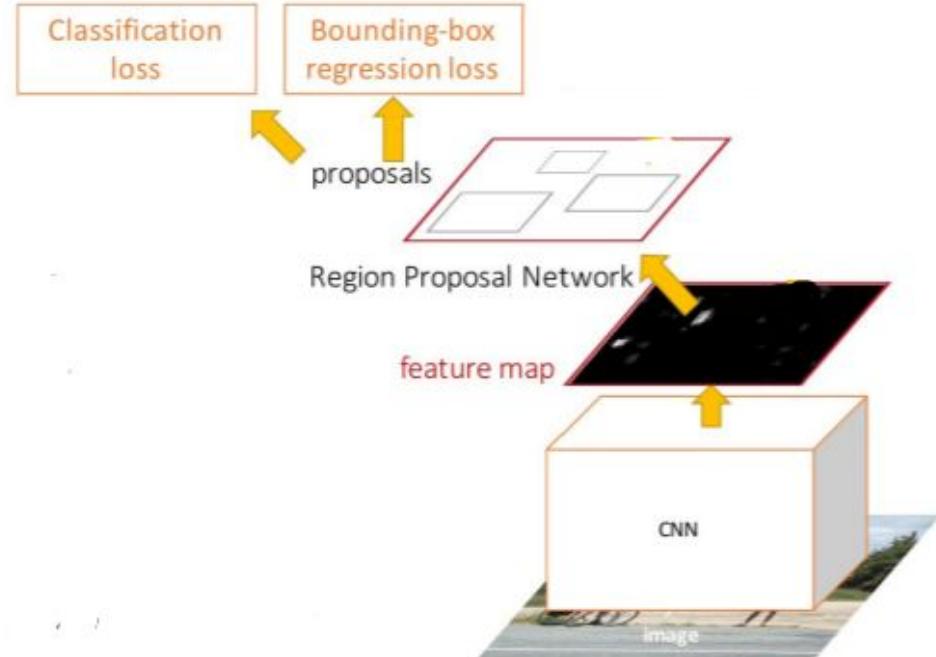
- RPN: классификация (объект/не объект)
- RPN: регрессия (координаты)
- Fast R-CNN: классификация (объект/не объект)
- Fast R-CNN: регрессия (координаты)



Детекция. Faster R-CNN

4-step alternating training:

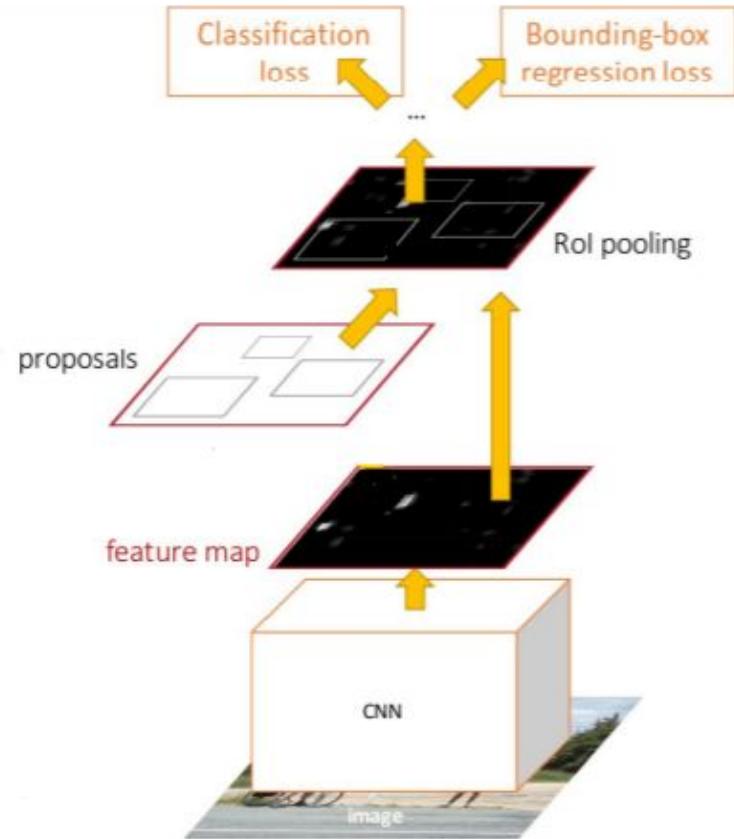
1. Обучаем RPN (предобученная CNN для инициализации)
2. Обучаем Fast R-CNN (предобученная CNN для инициализации), используем ROI из шага 1
3. Фиксируем CNN, обучаем RPN, не изменяем общие слои
4. Фиксируем CNN, обучаем Fast R-CNN, не изменяем общие слои



Детекция. Faster R-CNN

4-step alternating training:

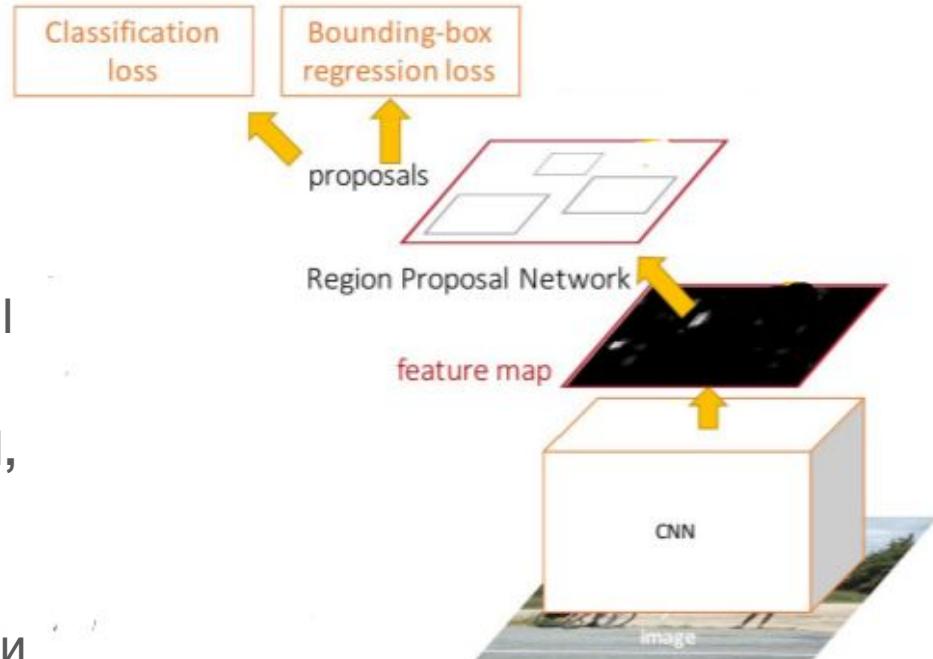
1. Обучаем RPN (предобученная CNN для инициализации)
2. Обучаем Fast R-CNN (предобученная CNN для инициализации), используем ROI из шага 1
3. Фиксируем CNN, обучаем RPN, не изменяем общие слои
4. Фиксируем CNN, обучаем Fast R-CNN, не изменяем общие слои



Детекция. Faster R-CNN

4-step alternating training:

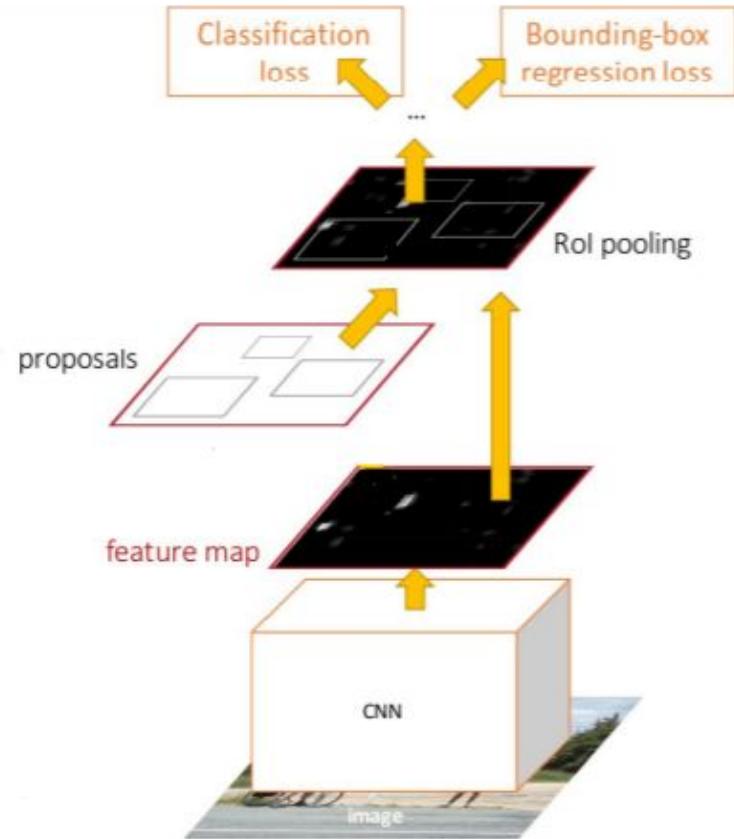
1. Обучаем RPN (предобученная CNN для инициализации)
2. Обучаем Fast R-CNN (предобученная CNN для инициализации), используем ROI из шага 1
3. **Фиксируем CNN, обучаем RPN, не изменяя общие слои**
4. Фиксируем CNN, обучаем Fast R-CNN, не изменяя общие слои



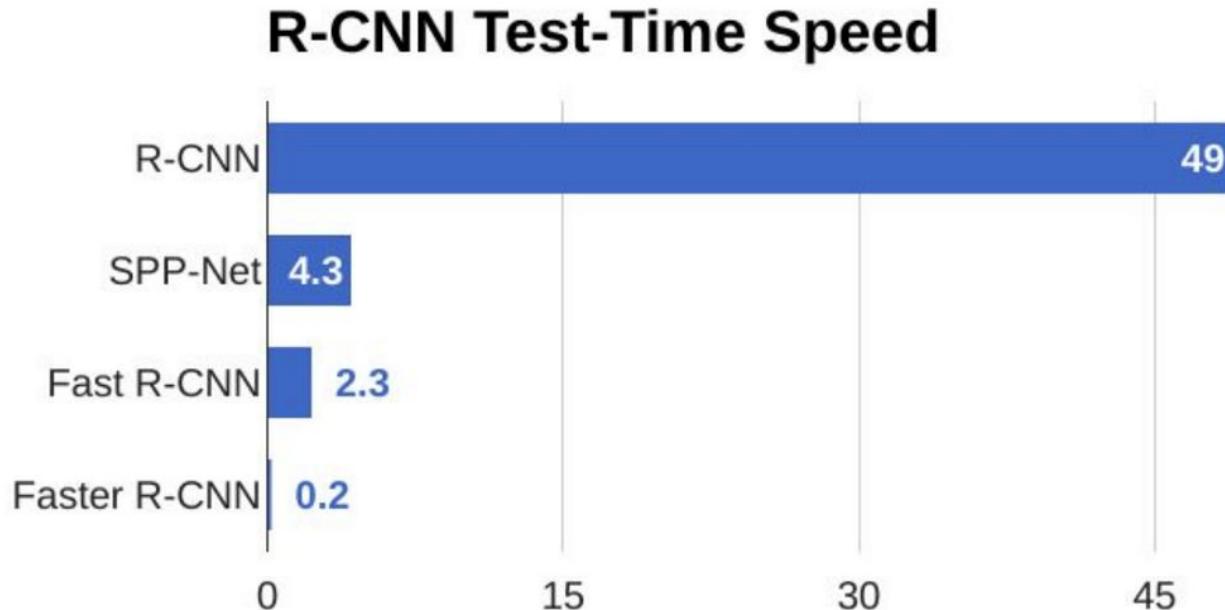
Детекция. Faster R-CNN

4-step alternating training:

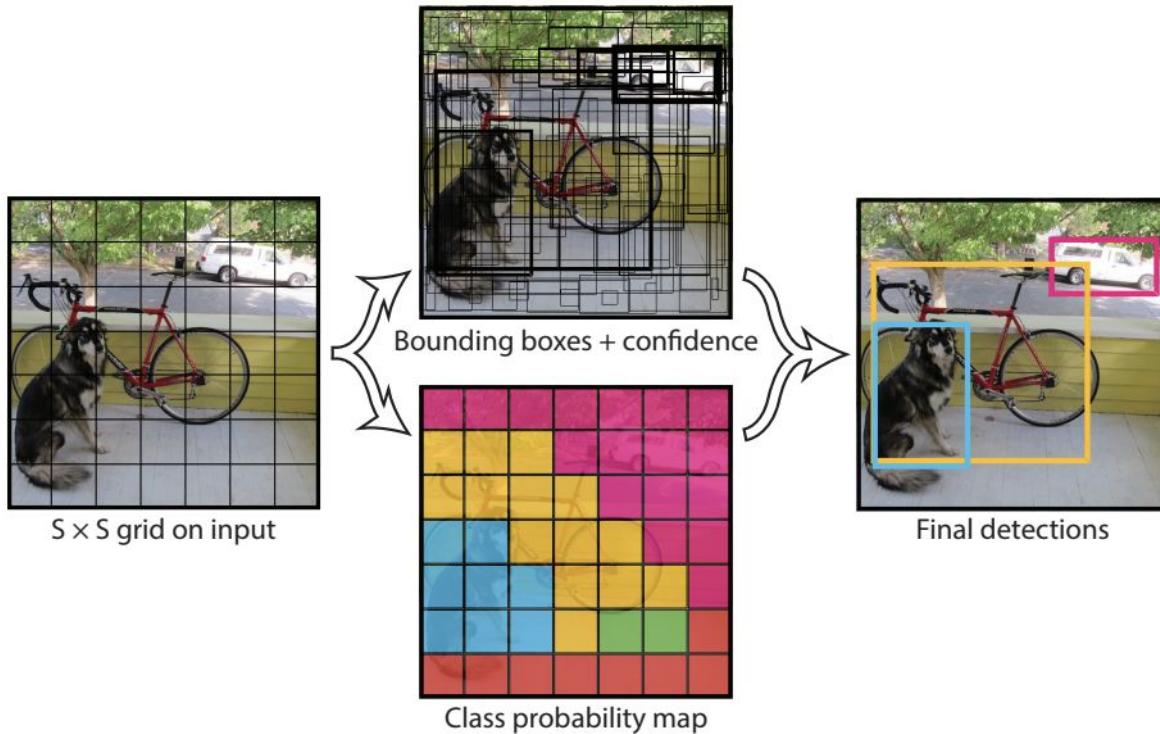
1. Обучаем RPN (предобученная CNN для инициализации)
2. Обучаем Fast R-CNN (предобученная CNN для инициализации), используем RoI из шага 1
3. Фиксируем CNN, обучаем RPN, не изменяем общие слои
4. Фиксируем CNN, обучаем Fast R-CNN, не изменяем общие слои



Детекция. R-CNN модели



Детекция. YOLO



YOLO:

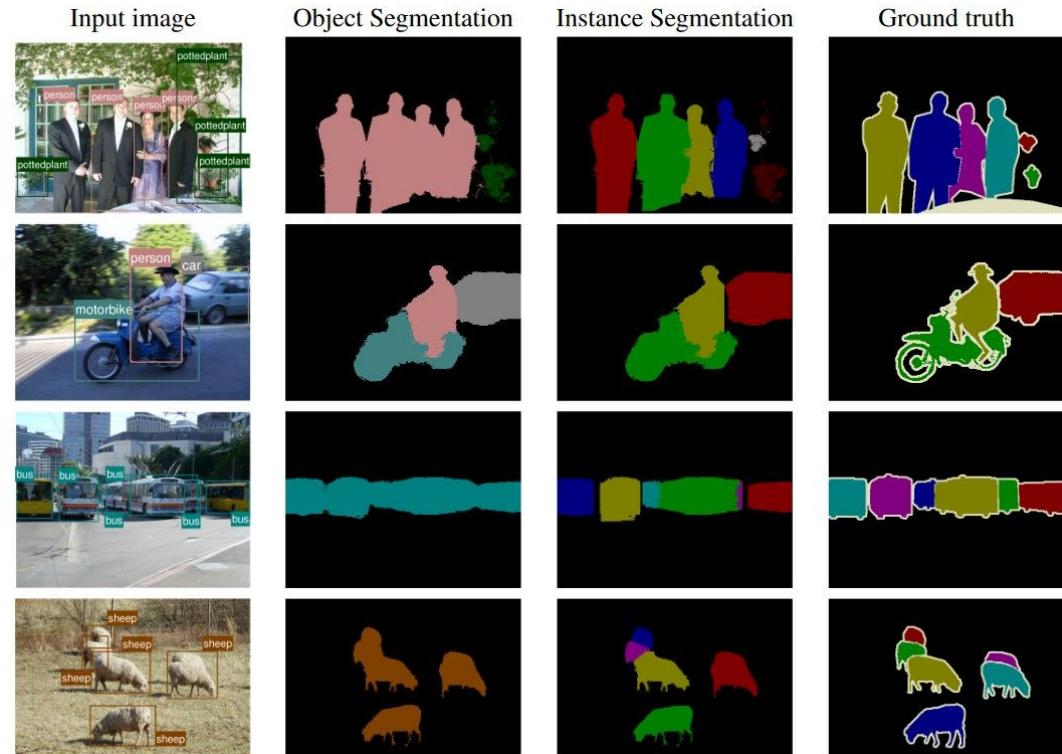
- В каждой ячейке рассматриваем B областей
- Регрессия ($x, y, w, h, confidence$)
- Классификация в каждой ячейке (из C классов)

Output: $S \times S \times (B \times 5 + C)$

Сегментация

- Semantic segmentation:
определение класса для
каждого пикселя
- Instance segmentation:
выделение отдельных
объектов внутри одного
класса

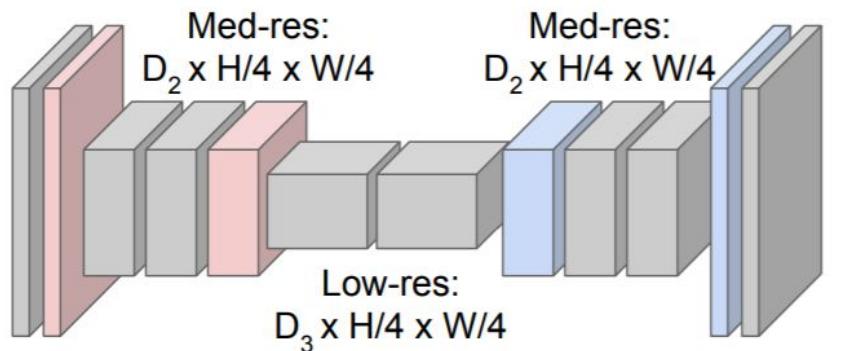
Instance segmentation =
Semantic segmentation +
Object detection



Semantic segmentation. FCN



Input:
 $3 \times H \times W$



High-res:
 $D_1 \times H/2 \times W/2$

High-res:
 $D_1 \times H/2 \times W/2$

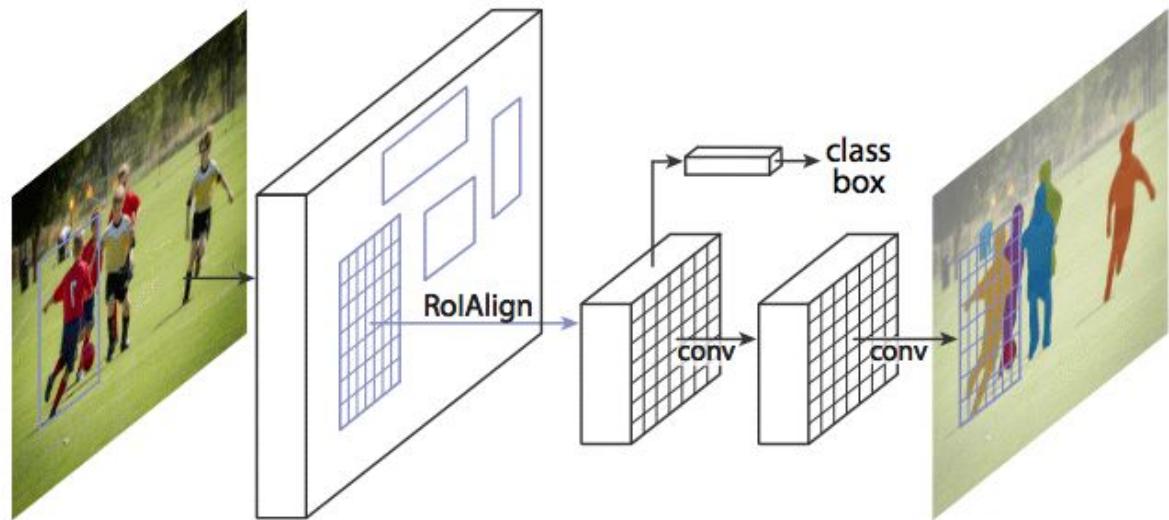


Predictions:
 $H \times W$

Instance segmentation. Mask R-CNN

Mask R-CNN:

к Faster R-CNN
добавляем сеть для
нахождения маски в
каждой области



$$L = L_{cls} + L_{bbox} + L_{mask}$$

Instance segmentation. Mask R-CNN



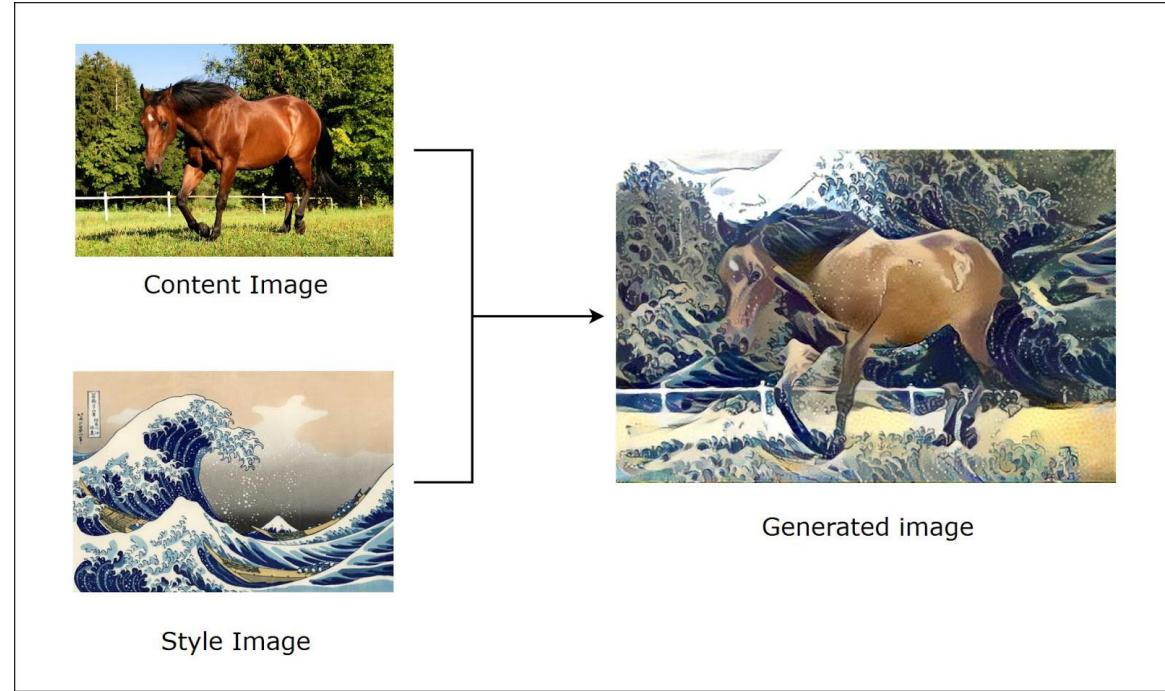
Instance segmentation. Mask R-CNN



Перенос стиля

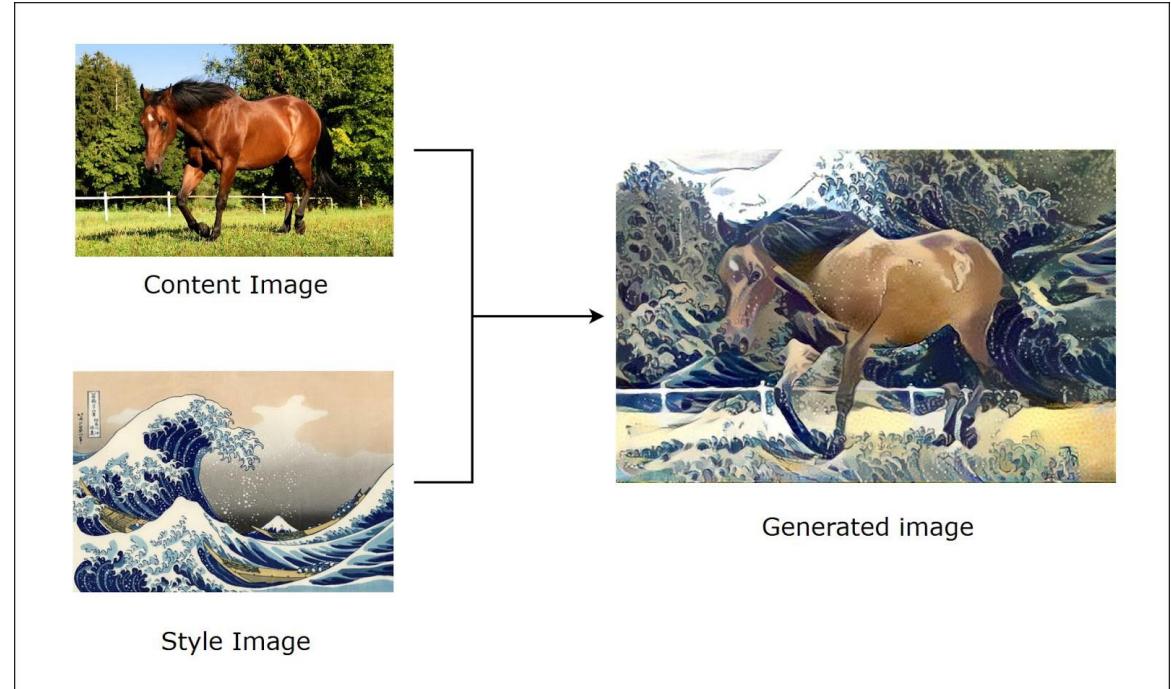


Перенос стиля



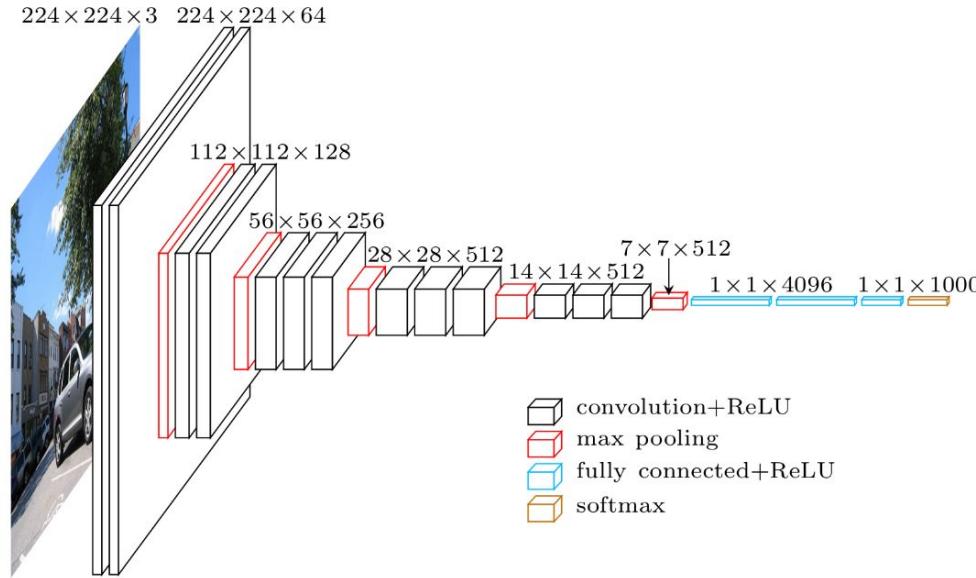
Задача – по 2 изображениям, содержащим контент и стиль соответственно, построить новое изображение, содержащее контент первого изображения и имеющее стиль второго.

Перенос стиля



Основная идея состоит в том, чтобы использовать CNN для извлечения признаков контента и стиля для данных изображений.

Алгоритм Neural Style Transfer



Возьмём сверточную часть обученной CNN для классификации изображений, например VGG-16. Мы используем её для получения признаков из изображений стиля и контента.

Алгоритм Neural Style Transfer

Определим функцию потерь следующим образом:

$$\mathcal{L}_{total}(C, S, G) = \alpha \mathcal{L}_C(C, G) + \beta \mathcal{L}_S(S, G)$$

где:

C – изображение контента

S – изображение стиля

G – изображение, которое мы хотим получить

\mathcal{L}_C – потеря контента

\mathcal{L}_S – потеря стиля

Алгоритм Neural Style Transfer

Определим потерю контента:

$$\mathcal{L}_C(C, G) = \frac{1}{2} \sum_{i,j,k} (C_{i,j,k}^l - G_{i,j,k}^l)^2$$

где C^l, G^l – выходы l -ого слоя CNN для изображений C и G

Алгоритм Neural Style Transfer

Определим потерю стиля:

Для каждого слоя CNN посчитаем матрицу корреляций между картами признаков этого слоя:

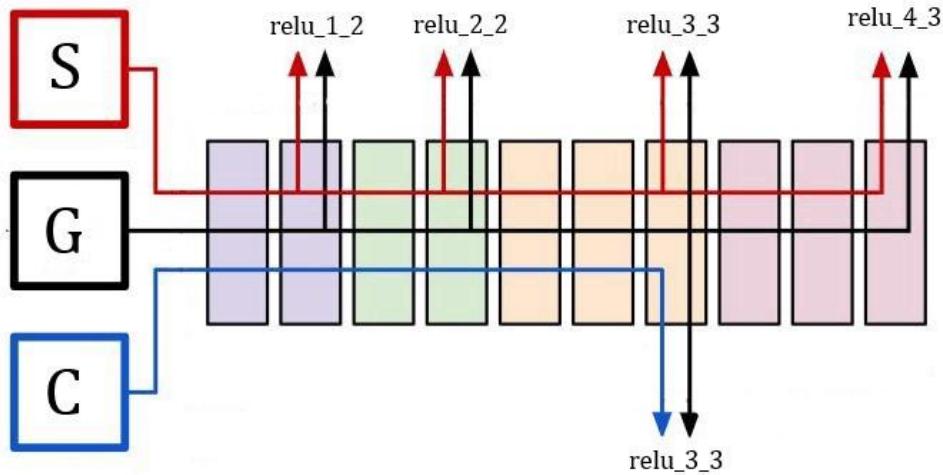
$$R^l(I)_{i,j} = \sum_{p,q} (F_i^l)_{p,q} (F_j^l)_{p,q}$$

где F_i^l – i-ая карта признаков l-ого слоя CNN при изображении I на входе.

Тогда потеря стиля:

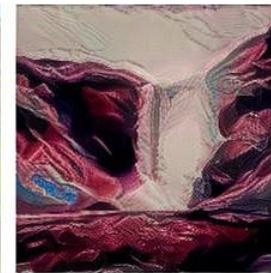
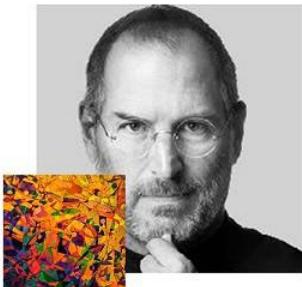
$$\mathcal{L}_S = \sum_{l=0}^L \frac{w_l}{(2S_l)^2} \sum_{i,j} (R^l(G)_{i,j} - R^l(S)_{i,j})^2$$
 где S_l – кол-во элементов в выходе l-го слоя, w_l – гиперпараметры

Алгоритм Neural Style Transfer



Инициализируем **G** нормальным шумом и оптимизируем функцию потерь с помощью градиентного спуска.

Результаты



Достоинства:

- 1) В отличие от донейросетевых подходов не имеет ограничения на переносимый стиль
- 2) Возможность регулировать соотношение стиль/контент в полученном изображении

Недостатки:

- 1) Большое время работы – мы каждый раз заново обучаем модель для получения изображения (для примера, генеративные модели для переноса стиля работают на порядок быстрее)

Super resolution



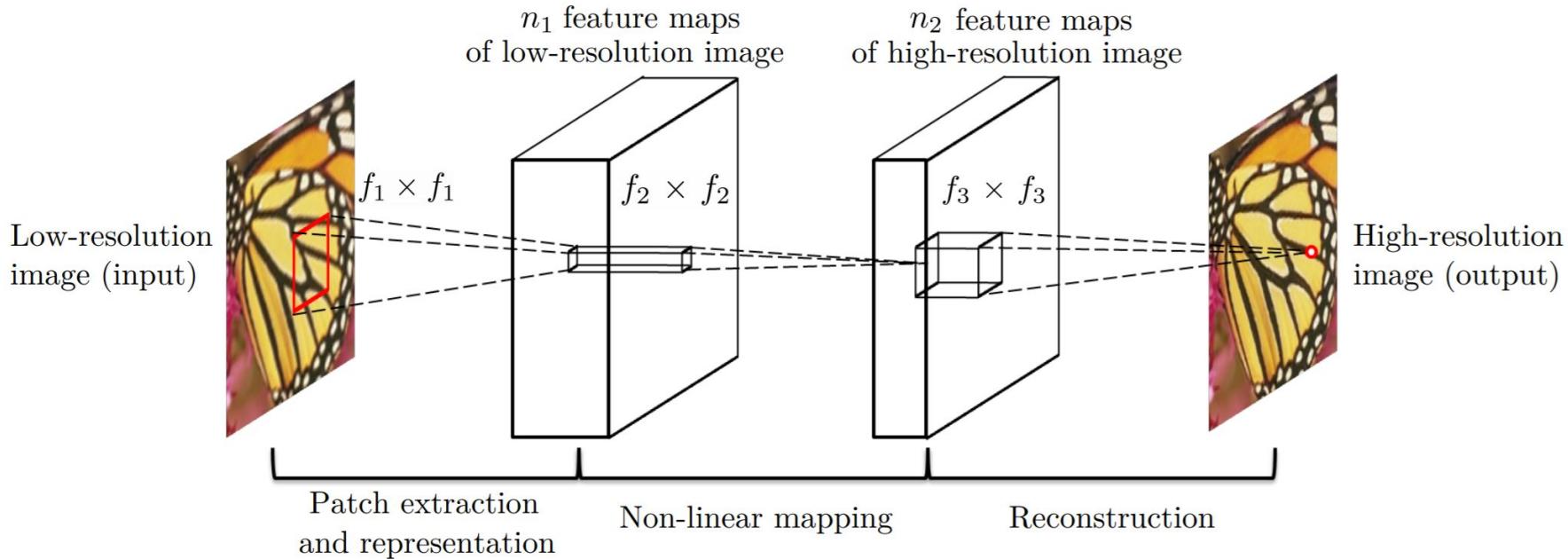
Задача — по изображению в низком разрешении построить изображение в высоком разрешении с восстановлением недостающих деталей.

SRCNN (Super Resolution CNN)

Одно из первых применений deep learning для решения задачи super resolution.

SRCNN по сложности не превосходит донейросетевые методы, применявшиеся для решения этой задачи, но выигрывает по качеству.

Архитектура SRCNN

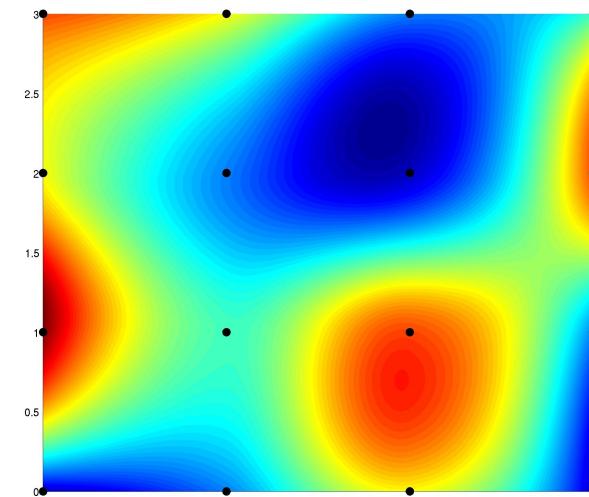


Предобработка изображения

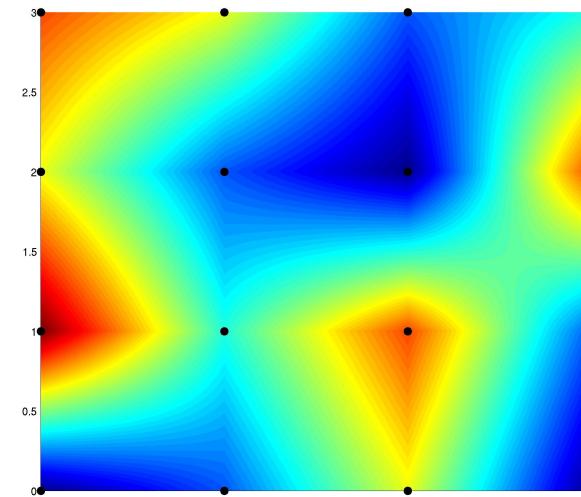
SRCNN получает на выходе изображение такого же размера, что и получает на вход. Значит нам нужно как-то увеличить разрешение изображения на этапе предобработки. Для этого мы используем интерполяцию.

Бикубическая интерполяция

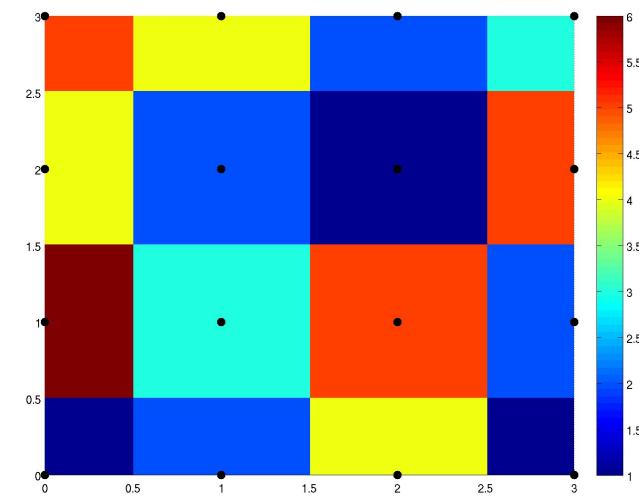
Зная значения функции в узлах сетки, мы хотим восстановить функцию на плоскости. Для этого приблизим её кубическим многочленом, коэффициенты которого вычислим по значениям функции в 16 ближайших точках.



Бикубическая интерполяция

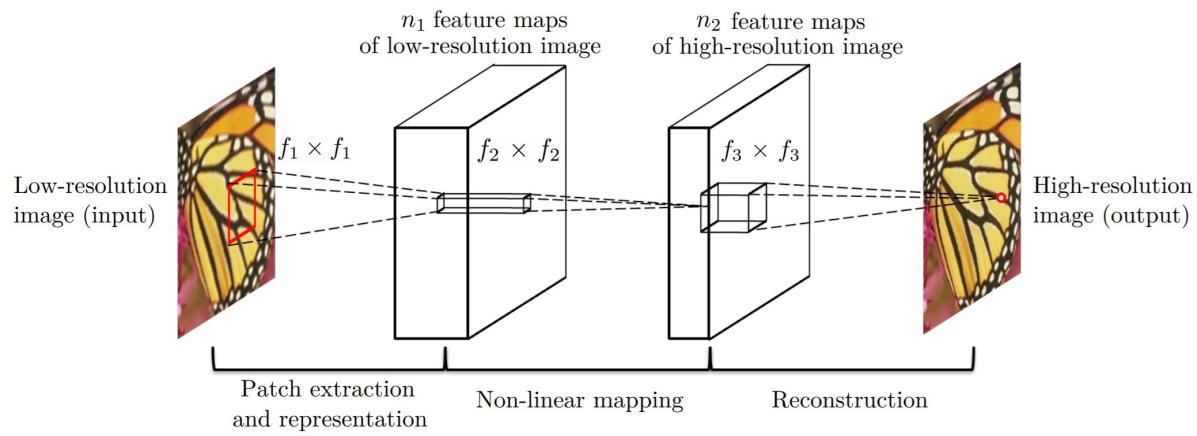


Билинейная интерполяция



Интерполяция к ближайшему соседу

Архитектура SRCNN



1-ый слой:

свертка с n_1 фильтрами
размера $c_{in} \times f_1 \times f_1 + \text{ReLU}$,
где c_{in} – число каналов в
исходном изображении

2-ой слой:

свертка с n_2 фильтрами
размера $n_1 \times f_2 \times f_2 + \text{ReLU}$

3-ий слой:

свертка с c_{in} фильтрами
размера $n_2 \times f_3 \times f_3$

В качестве функции потерь используется MSE:

$$L(\Theta) = \frac{1}{n} \sum_{i=1}^n \|F(\mathbf{Y}_i; \Theta) - \mathbf{X}_i\|^2$$

где \mathbf{X}_i – изображение в высоком разрешении, \mathbf{Y}_i – вход модели, а Θ – параметры модели.

Оптимизируется функция потерь с помощью Momentum:

$$\Delta_{i+1} = 0.9 \cdot \Delta_i - \eta \cdot \frac{\partial L}{\partial W_i^\ell}, \quad W_{i+1}^\ell = W_i^\ell + \Delta_{i+1}$$

Метрики качества

PSNR – peak signal-to-noise ratio:

$$PSNR(I, K) = 10 \log_{10} \left(\frac{MAX_I^2}{MSE(I, K)} \right) = 20 \log_{10} \left(\frac{MAX_I}{\sqrt{MSE(I, K)}} \right)$$

$$MSE(I, K) = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |I(i, j) - K(i, j)|^2 , \text{ где } I \text{ и } K \text{ – изображения } m \times n$$

Метрики качества

SSIM – structural similarity index measure:

$$\text{SSIM}(I, K) = \frac{(2\mu_I\mu_K + c_1)(2\sigma_{IK} + c_2)}{(\mu_I^2 + \mu_K^2 + c_1)(\sigma_I^2 + \sigma_K^2 + c_2)}, \text{ где:}$$

μ_I, μ_K — среднее изображений I и K соответственно

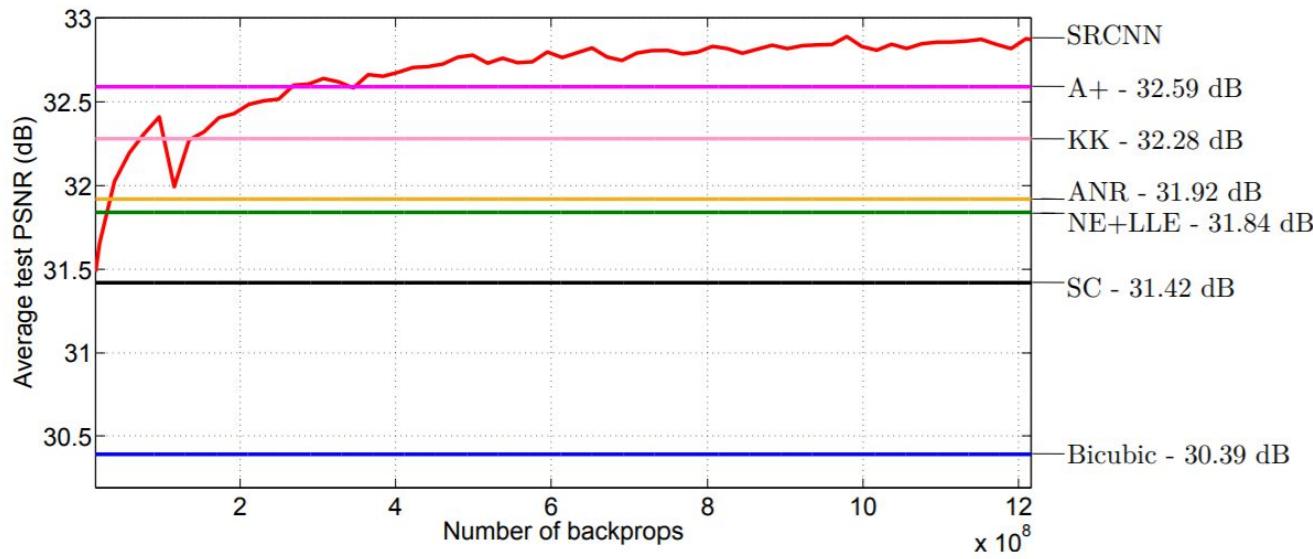
σ_I, σ_K — дисперсия I и K

σ_{IK} — ковариация I и K

c_1, c_2 — константы

$\text{SSIM} \in [-1, 1]$, $\text{SSIM} = 1 \Leftrightarrow I = K$

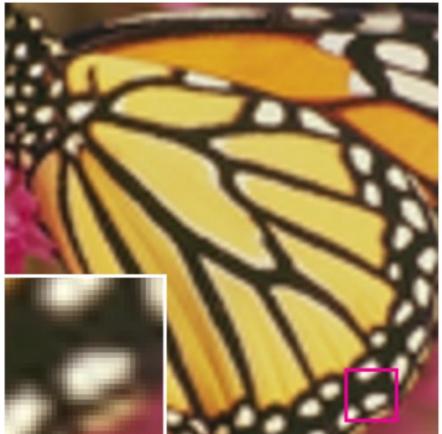
Результаты:



Eval. Mat	Scale	Bicubic	SC [50]	NE+LLE [4]	KK [25]	ANR [41]	A+ [41]	SRCNN
PSNR	2	28.38	-	29.67	30.02	29.72	30.14	30.29
	3	25.94	26.54	26.67	26.89	26.72	27.05	27.18
	4	24.65	-	25.21	25.38	25.25	25.51	25.60
SSIM	2	0.8524	-	0.8886	0.8935	0.8900	0.8966	0.8977
	3	0.7469	0.7729	0.7823	0.7881	0.7843	0.7945	0.7971
	4	0.6727	-	0.7037	0.7093	0.7060	0.7171	0.7184



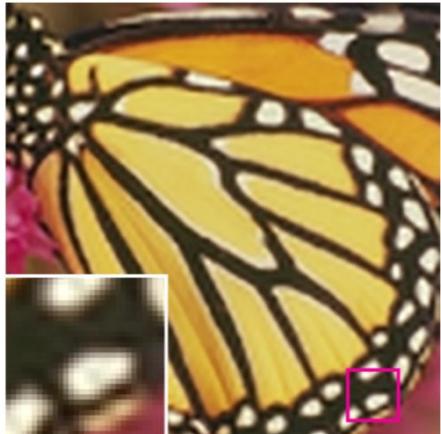
Original / PSNR



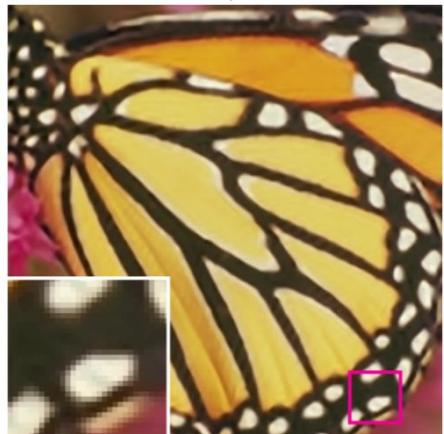
Bicubic / 24.04 dB



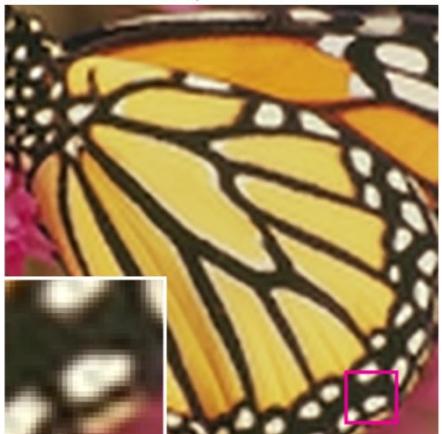
SC / 25.58 dB



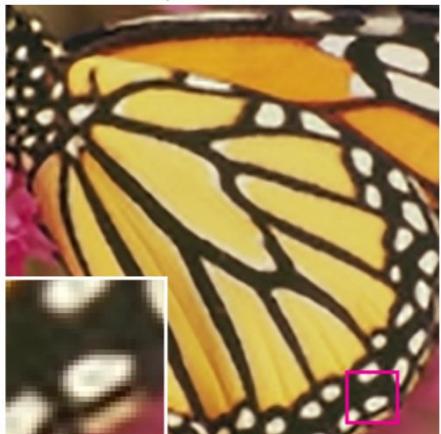
NE+LLE / 25.75 dB



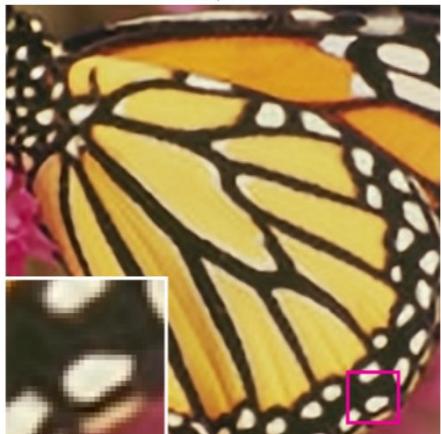
KK / 27.31 dB



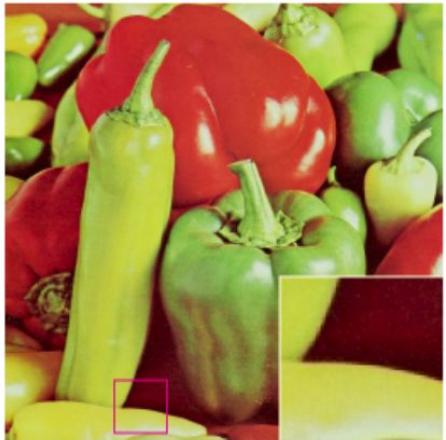
ANR / 25.90 dB



A+ / 27.24 dB



SRCNN / **27.95 dB**



Original / PSNR



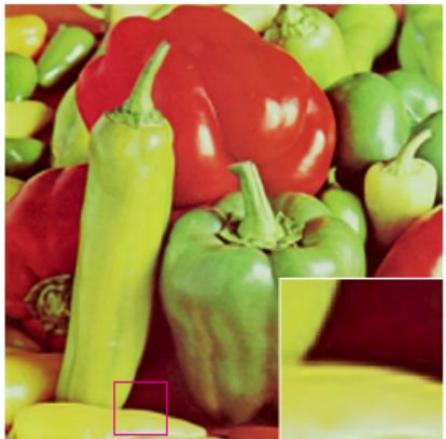
Bicubic / 32.39 dB



SC / 33.32 dB



K-SVD / 34.07 dB



NE+NNLS / 33.56 dB



NE+LLE / 33.80 dB



ANR / 33.82 dB



SRCCN / 34.35 dB

Достоинства:

- Простота архитектуры
- Возможность обработки всех каналов цветных изображений одновременно

Недостатки:

- Все вычисления производятся с изображением в высоком разрешении

SRDenseNet

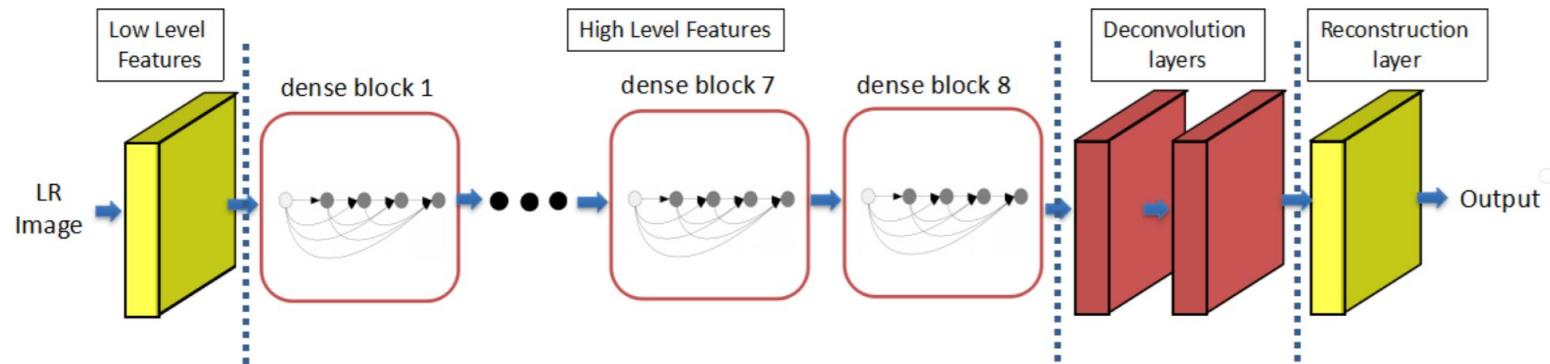
Адаптация идеи DenseNet к задаче SR.

Мотивация:

В предыдущих моделях изображение приводилось к нужному разрешению на этапе предобработки.

Вычислительно проще будет оперировать изображением в низком разрешении и приводить его к нужному разрешению на конечных этапах модели.

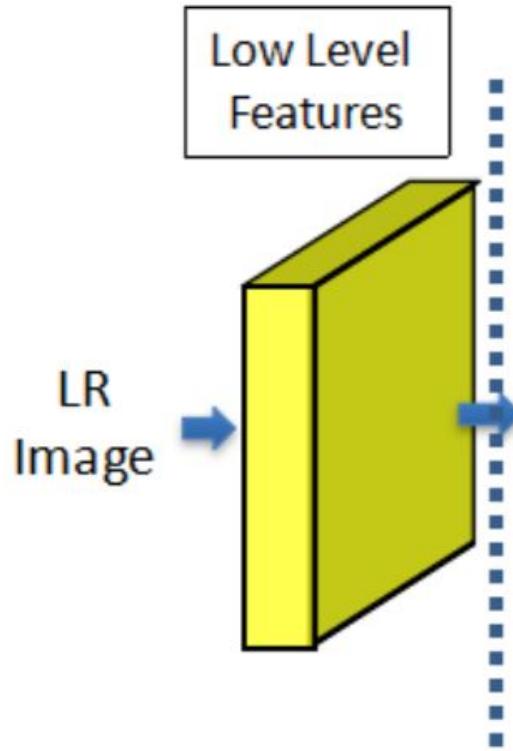
Архитектура SRDenseNet



Архитектура SRDenseNet

1) Свертка + ReLU

Служит для выделения низкоуровневых признаков на изображении



Архитектура SRDenseNet

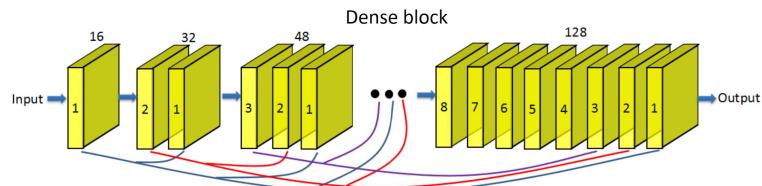
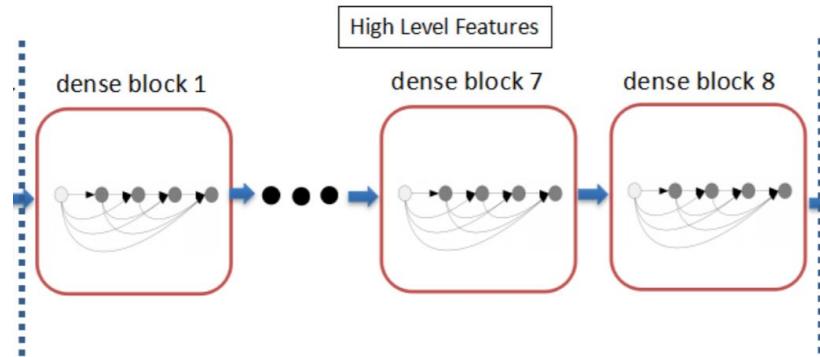
1) Свертка + ReLU

Служит для выделения низкоуровневых признаков на изображении

2) 8 dense блоков

Каждый dense блок: 8 сверток с конкатенацией предыдущих слоев + ReLU

Dense блоки служат для выделения высокоуровневых признаков



Архитектура SRDenseNet

1) Свертка + ReLU

Служит для выделения низкоуровневых признаков на изображении

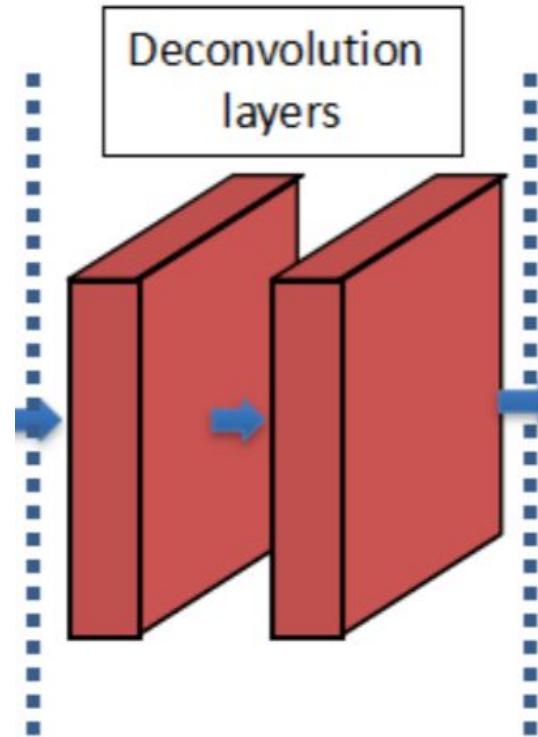
2) 8 dense блоков

Каждый dense блок: 8 сверток с конкатенацией предыдущих слоев + ReLU

Dense блоки служат для выделения высокоуровневых признаков

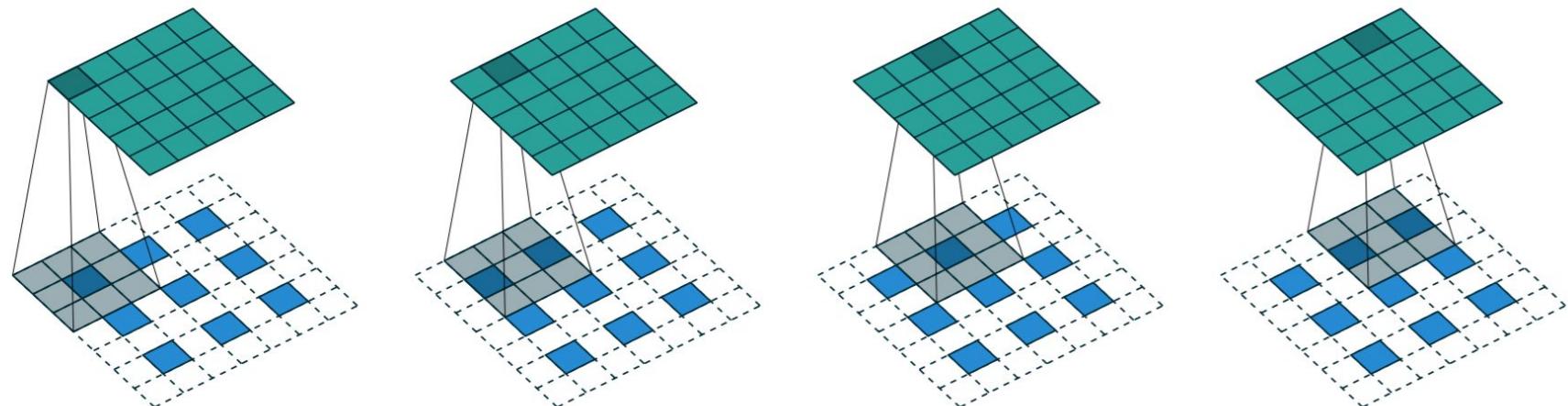
3) Deconvolution layer + ReLU:

Используется для приведения изображения к нужному разрешению.



Deconvolution

Deconvolution или transposed convolution – операция, обратная свертке.



В отличие от SRCNN, где для увеличения используется интерполяция, в SRDenseNet для этого используется транспонированная свертка.

Архитектура SRDenseNet

1) Свертка + ReLU

Служит для выделения низкоуровневых признаков на изображении

2) 8 dense блоков

Каждый dense блок: 8 сверточ с конкатенацией предыдущих слоев + ReLU

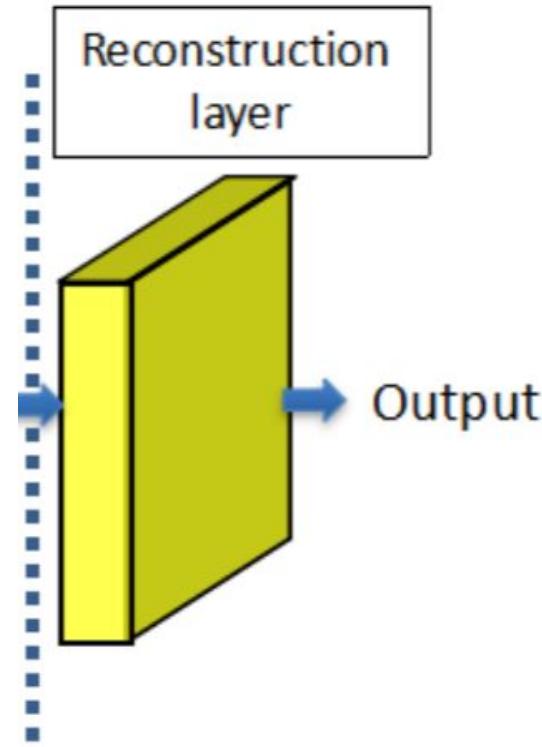
Dense блоки служат для выделения высокоуровневых признаков

3) Deconvolution layer + ReLU:

Используется для приведения изображения к нужному разрешению.

4) Reconstruction layer:

Свертка для получения конечного результата

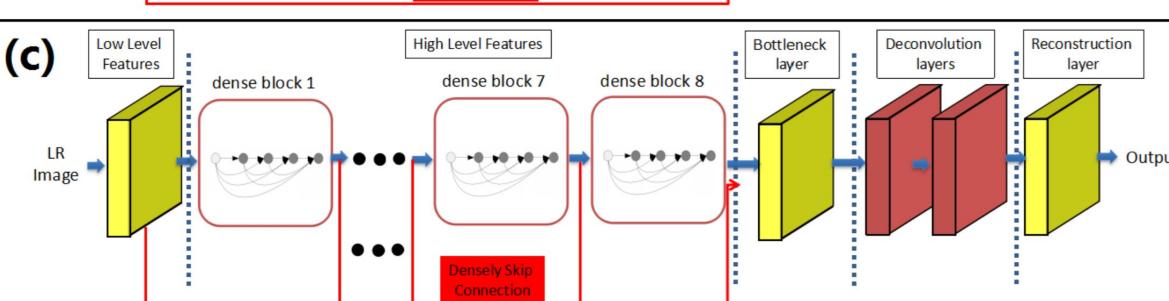
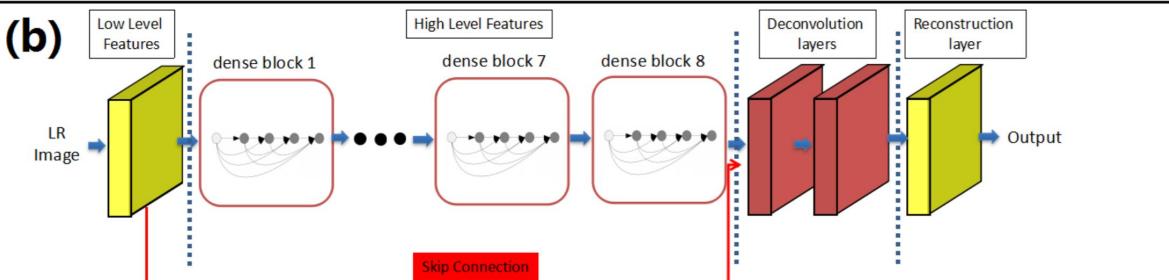
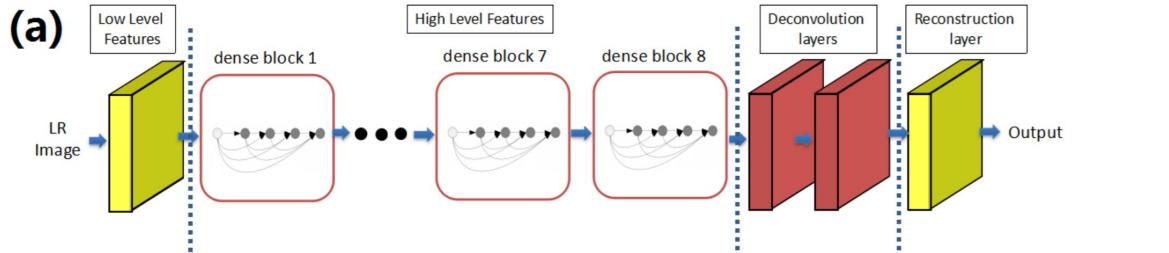


Архитектура SRDenseNet

Детали:

- 1) SRDenseNet не работает со всеми каналами цветных изображений. Если исходное изображение в RGB, то оно преобразуется в YCbCr, где Y отвечает за яркость, и модель работает только с ним, а каналы Cb и Cr апскейляются с помощью интерполяции.
- 2) Так же, как и в SRCNN, в качестве ошибки используется MSE, но оптимизируется она с помощью Adam.

Возможные вариации архитектуры



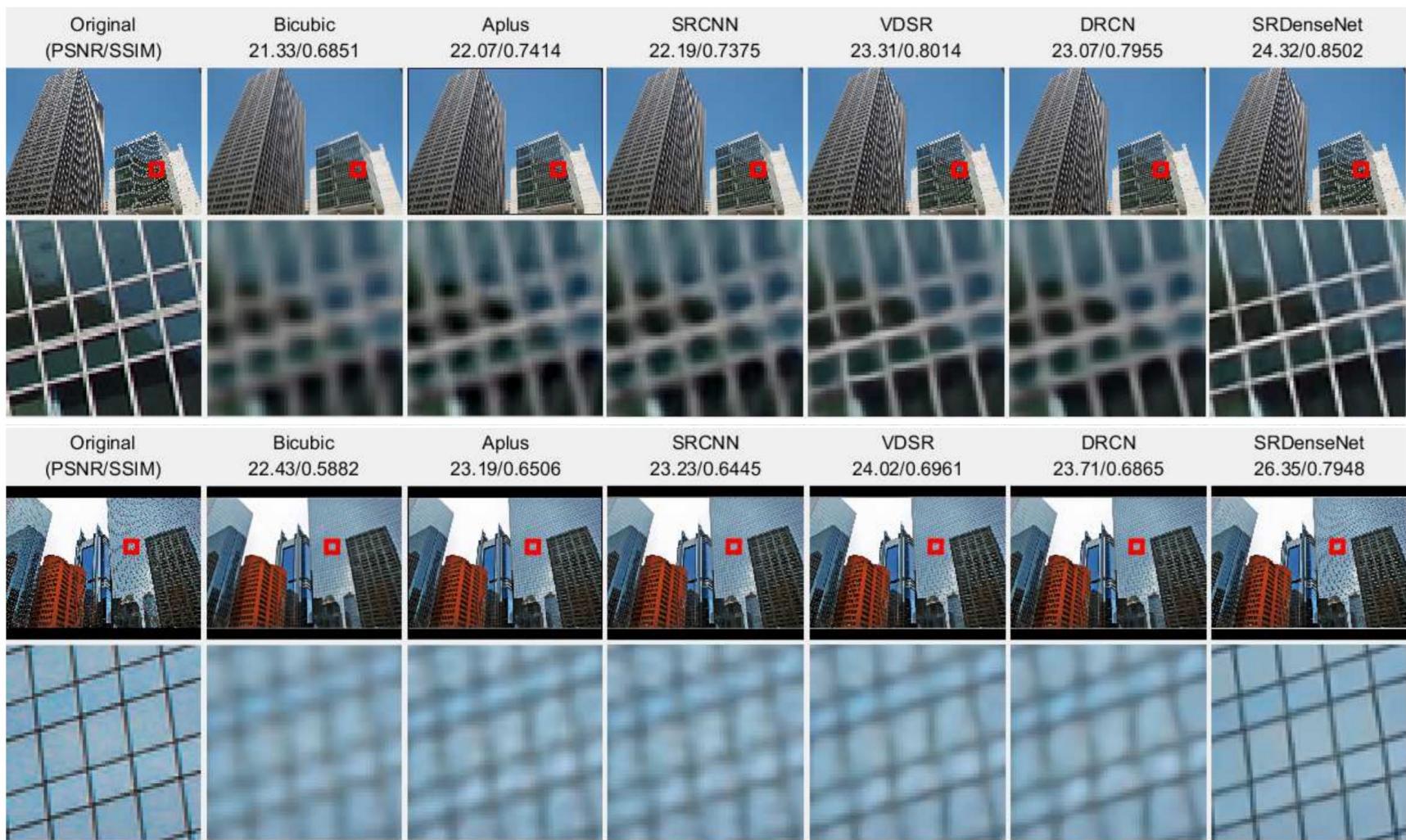
Результаты

Dataset	SRDenseNet_H	SRDenseNet_HL	SRDenseNet_All
Urban100	25.69/0.7700	25.86/0.7761	26.05/0.7819
Set5	31.66/0.8882	31.80/0.8907	32.02/0.8934
Set14	28.34/0.7744	28.40/0.7765	28.50/0.7782
B100	27.42/0.7300	27.47/0.7318	27.53/0.7337

Результаты

Dataset	SRDenseNet_H	SRDenseNet_HL	SRDenseNet_All
Urban100	25.69/0.7700	25.86/0.7761	26.05/0.7819
Set5	31.66/0.8882	31.80/0.8907	32.02/0.8934
Set14	28.34/0.7744	28.40/0.7765	28.50/0.7782
B100	27.42/0.7300	27.47/0.7318	27.53/0.7337

Dataset	Bicubic	Aplus [24]	SRCNN [2]	VDSR [11]	DRCN [12]	SRDenseNet_All
Urban100	23.14/0.6577	24.32/0.7183	24.52/0.7221	25.18/0.7524	25.14/0.7510	26.05/0.7819
Set5	28.42/0.8104	30.28/0.8603	30.48/0.8628	31.35/0.8838	31.53/0.8854	32.02/0.8934
Set14	26.00/0.7027	27.32/0.7491	27.49/0.7503	28.01/0.7674	28.02/0.7670	28.50/0.7782
B100	25.96/0.6675	26.82/0.7087	26.90/0.7101	27.29/0.7251	27.23/0.7233	27.53/0.7337
All	24.73/0.6685	25.79/0.7191	25.93/0.7216	26.47/0.7439	26.42/0.7424	27.02/0.7622



Original
(PSNR/SSIM)

Bicubic
22.03/0.8219

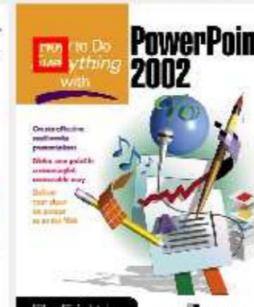
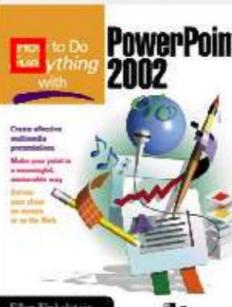
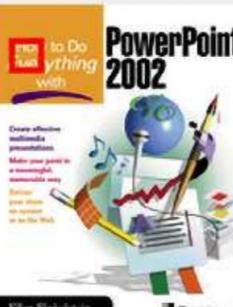
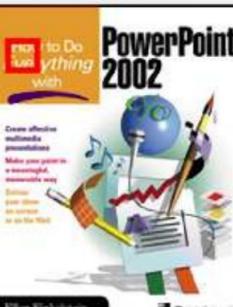
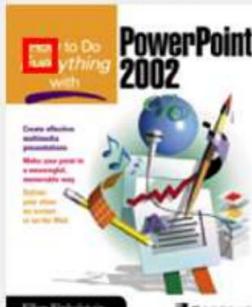
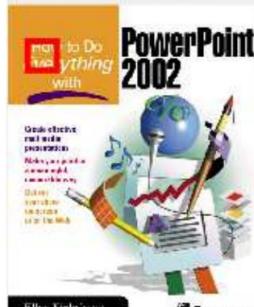
Aplus
23.72/0.8838

SRCCN
23.90/0.8741

VDSR
25.92/0.9306

DRCN
25.49/0.9262

SRDenseNet
28.01/0.9590



MS-LapSRN

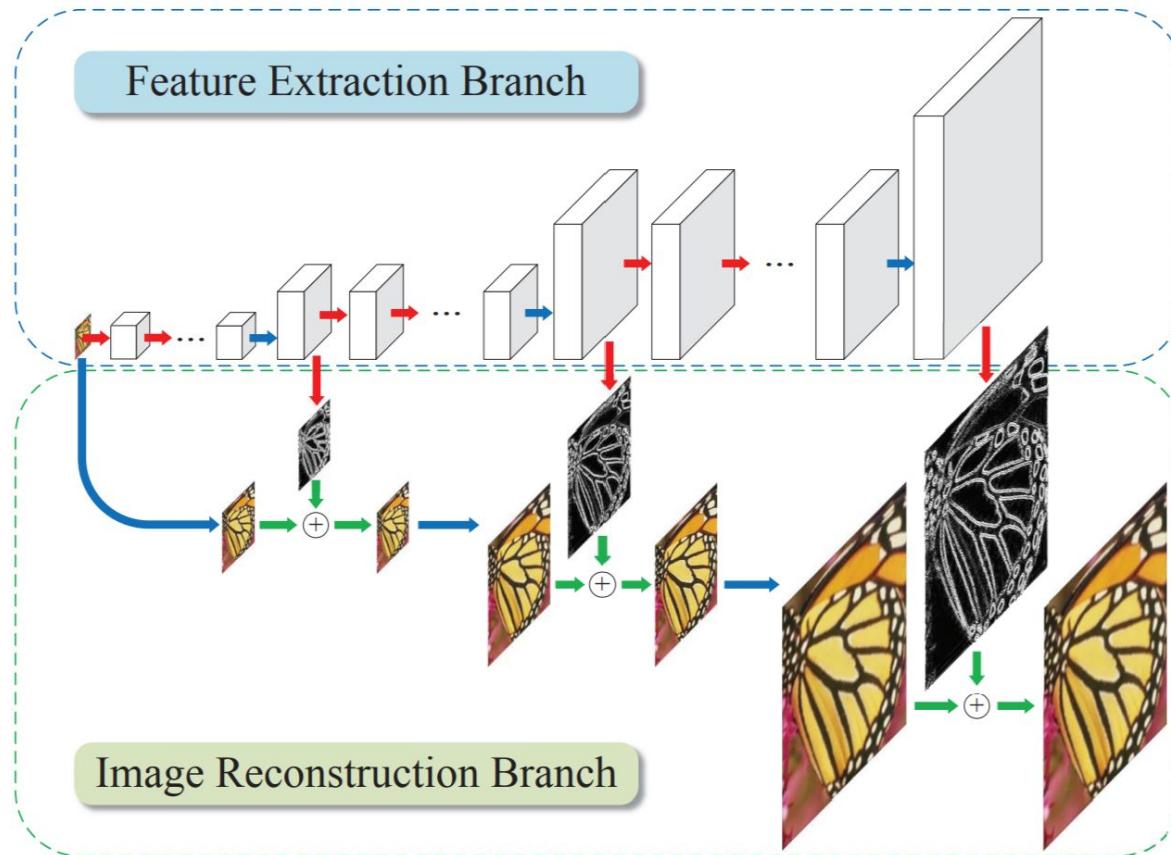
MS-LapSRN – Multiscale Laplacian Pyramid Super Resolution Network

Мотивация:

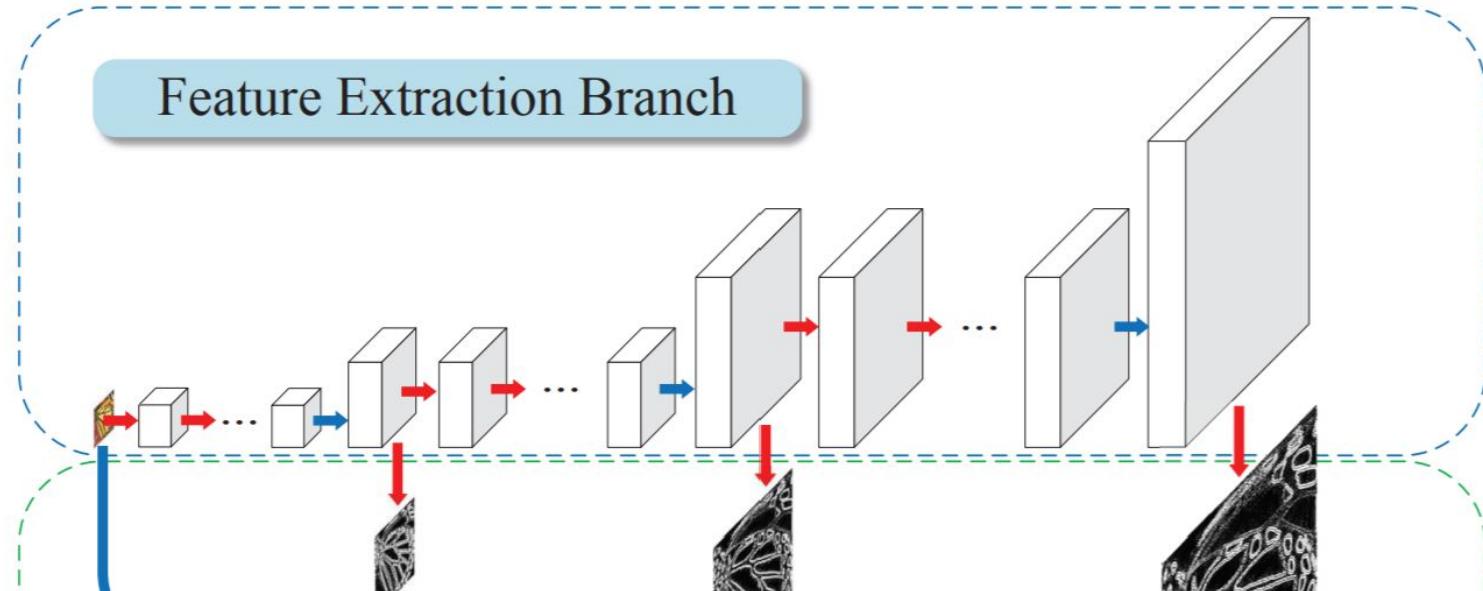
При большом увеличении разрешения (например x8 или больше) модели может быть трудно восстановить все детали за один этап.

Идея MS-LapSRN состоит в последовательном увеличении разрешения x2 несколько раз.

Архитектура MS-LapSRN

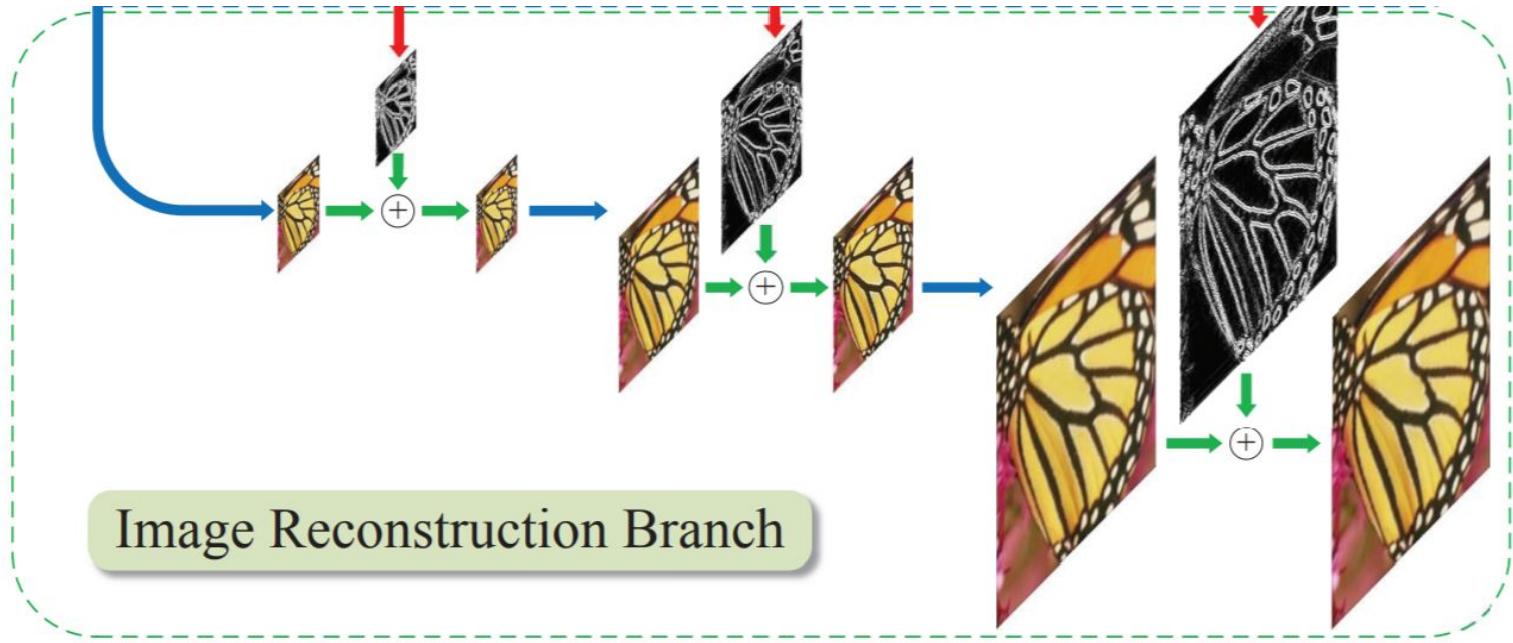


Архитектура MS-LapSRN



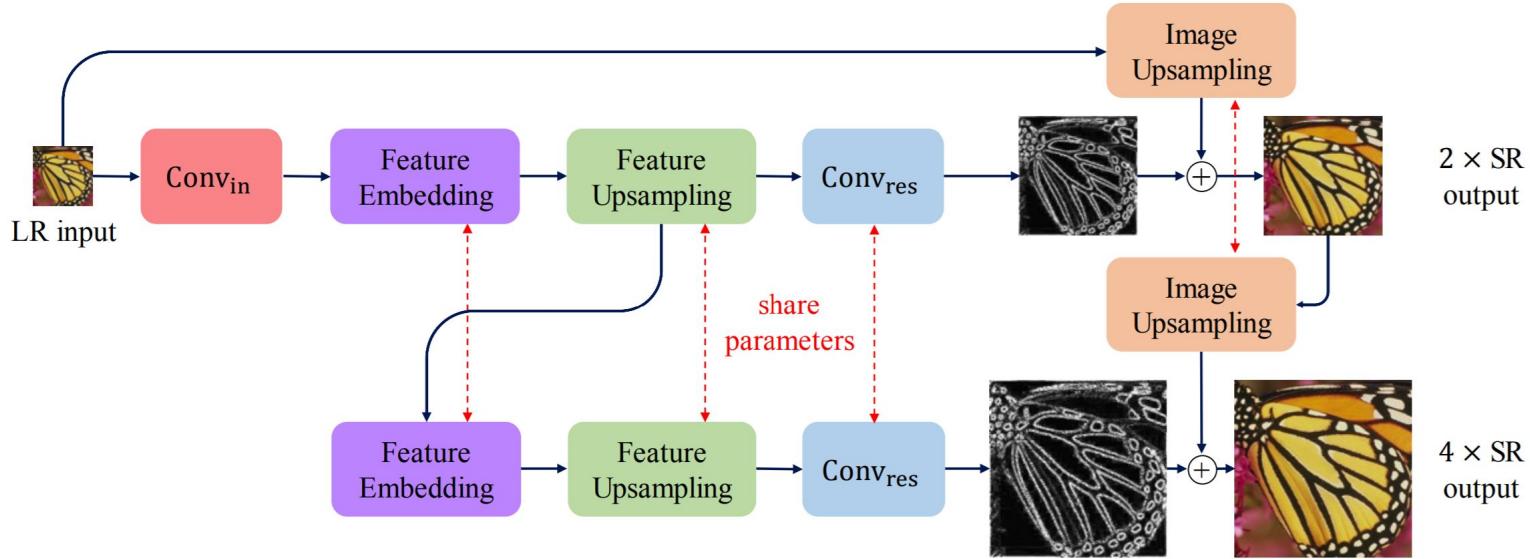
Каждый уровень в ветви выделения признаков – D conv. layers + 1 deconv. Layer
Выход каждого разверточного слоя также подаётся на вход сверточному слою,
который обучается восстанавливать остаточное изображение для подачи на ветвь
восстановления изображения.

Архитектура MS-LapSRN



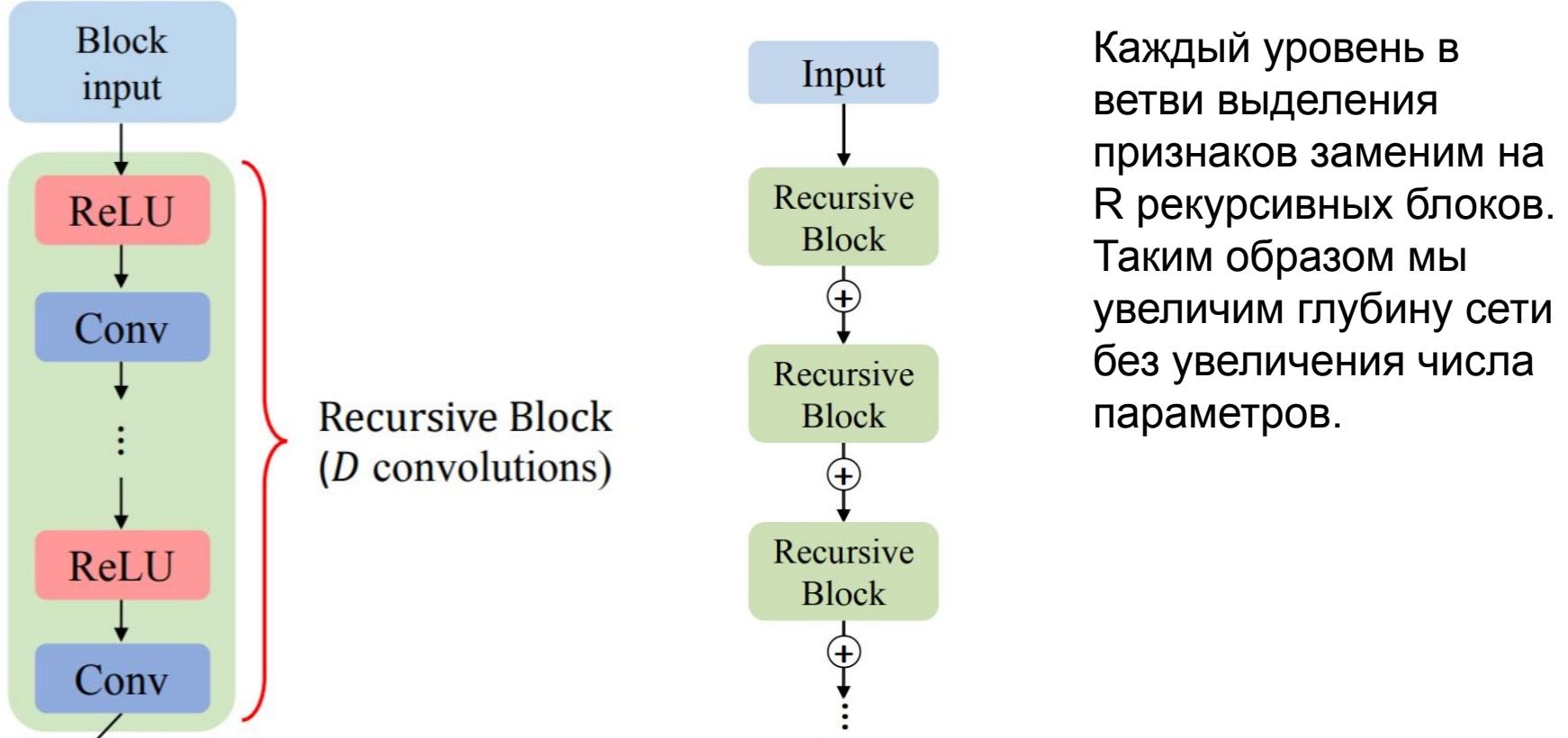
На каждом уровне текущее изображение складывается с предсказанным ветвью выделения признаков остаточным изображением, после чего поступает на вход разверточному слою.

Архитектура MS-LapSRN

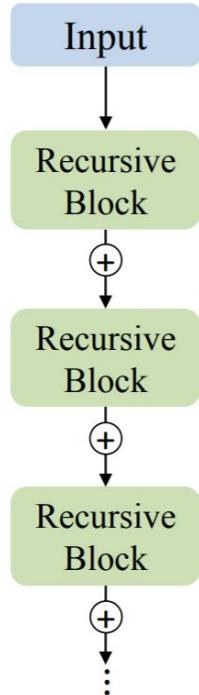


Для уменьшения кол-ва параметров будем переиспользовать одни и те же параметры на каждом уровне сети

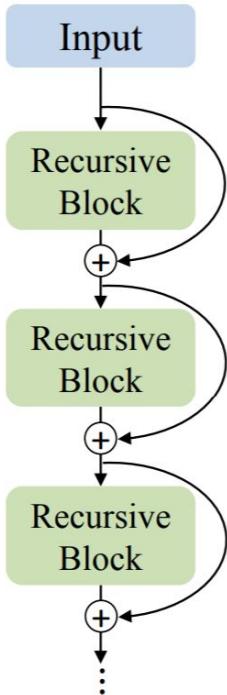
Архитектура MS-LapSRN



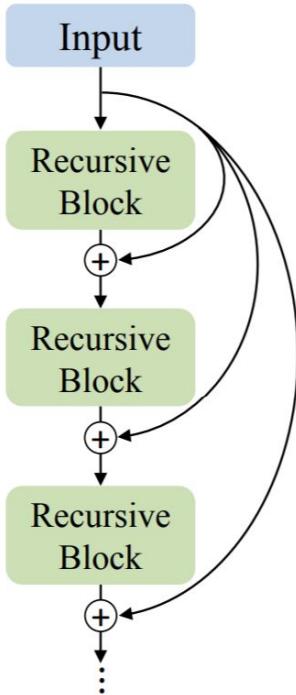
Архитектура MS-LapSRN



(a) No skip connection



(b) Distinct-source skip connection



(c) Shared-source skip connection

Local Residual Learning

(a) LapSRN_NS

(b) LapSRN_DS

(c) LapSRN_SS

Функция потерь

В качестве функции потерь используется Charbonnier loss:

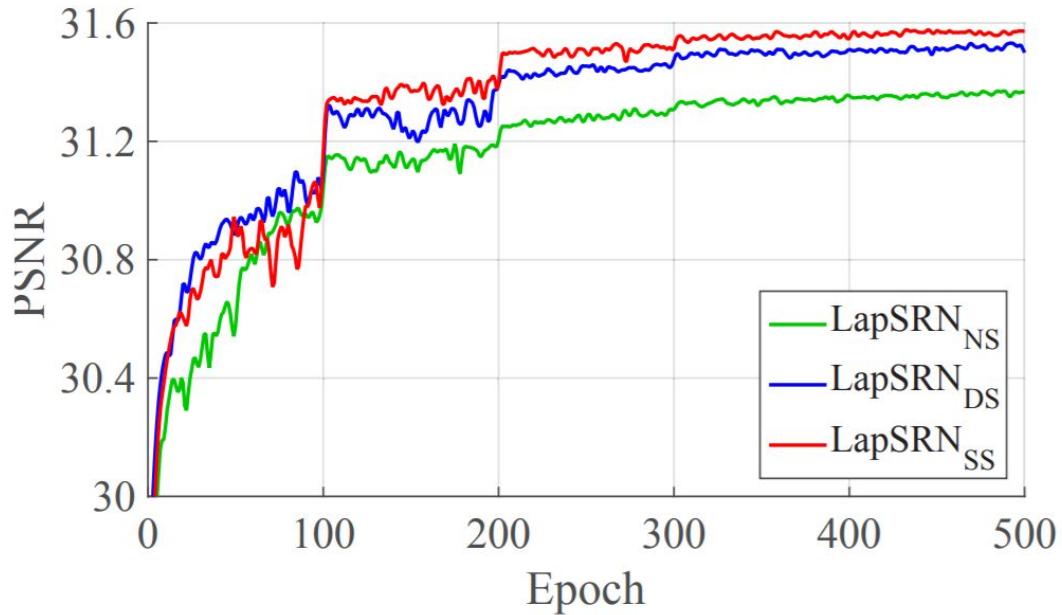
$$\mathcal{L}(y, \hat{y}; \theta) = \sum \mathcal{L}_S(y, \hat{y}; \theta)$$

$$\mathcal{L}_S(y, \hat{y}; \theta) = \frac{1}{N} \sum_{i=1}^N \sum_{l=1}^L \rho \left(y_l^{(i)} - \hat{y}_l^{(i)} \right), \text{ где:}$$

$$\rho(x) = \sqrt{(x^2 + \epsilon)}$$

\hat{y} – выход модели, N – кол-во примеров в батче, S – масштаб,
 L – кол-во уровней в сети

Результаты



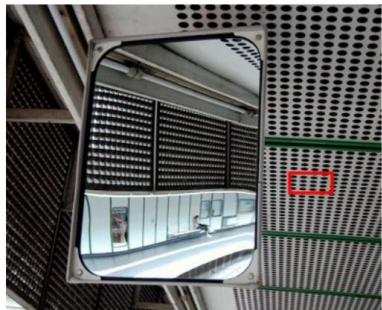
Model	Depth	LapSRN _{NS}	LapSRN _{DS}	LapSRN _{SS}
D5R2	24	25.16	25.22	25.23
D5R5	54	25.18	25.33	25.34
D5R8	84	25.26	25.33	25.38

Результаты

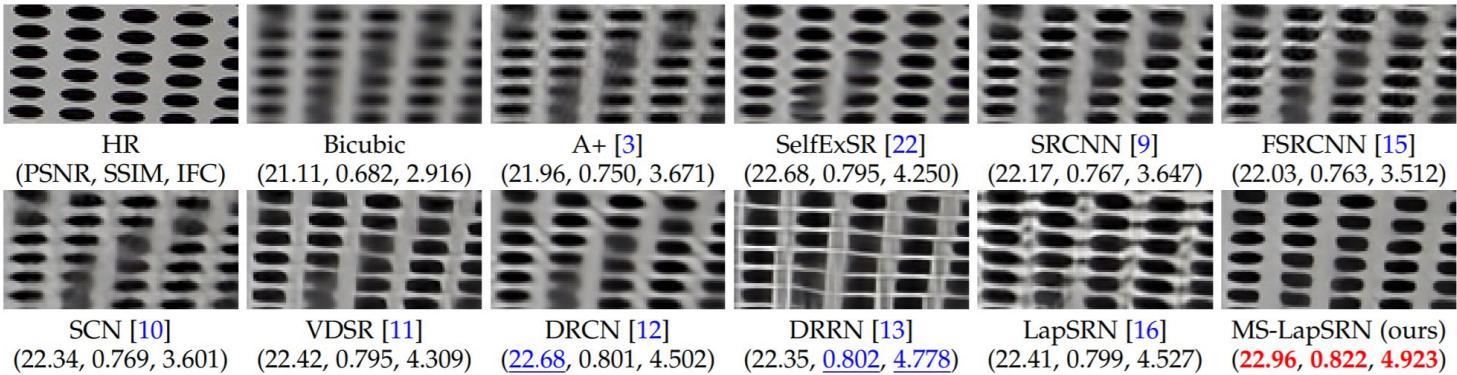
Algorithm	Scale	SET5			SET14			BSDS100			URBAN100			MANGA109		
		PSNR	SSIM	IFC	PSNR	SSIM	IFC	PSNR	SSIM	IFC	PSNR	SSIM	IFC	PSNR	SSIM	IFC
Bicubic	2×	33.69	0.931	6.166	30.25	0.870	6.126	29.57	0.844	5.695	26.89	0.841	6.319	30.86	0.936	6.214
A+ [3]		36.60	0.955	8.715	32.32	0.906	8.200	31.24	0.887	7.464	29.25	0.895	8.440	35.37	0.968	8.906
RFL [5]		36.59	0.954	8.741	32.29	0.905	8.224	31.18	0.885	7.473	29.14	0.891	8.439	35.12	0.966	8.921
SelfExSR [22]		36.60	0.955	8.404	32.24	0.904	8.018	31.20	0.887	7.239	29.55	0.898	8.414	35.82	0.969	8.721
SRCNN [9]		36.72	0.955	8.166	32.51	0.908	7.867	31.38	0.889	7.242	29.53	0.896	8.092	35.76	0.968	8.471
FSRCNN [15]		37.05	0.956	8.199	32.66	0.909	7.841	31.53	0.892	7.180	29.88	0.902	8.131	36.67	0.971	8.587
SCN [10]		36.58	0.954	7.358	32.35	0.905	7.085	31.26	0.885	6.500	29.52	0.897	7.324	35.51	0.967	7.601
VDSR [11]		37.53	0.959	8.190	33.05	0.913	7.878	31.90	0.896	7.169	30.77	0.914	8.270	37.22	0.975	9.120
DRCN [12]		37.63	0.959	8.326	33.06	0.912	8.025	31.85	0.895	7.220	30.76	0.914	8.527	37.63	0.974	9.541
LapSRN [16]		37.52	0.959	9.010	33.08	0.913	8.501	31.80	0.895	7.715	30.41	0.910	8.907	37.27	0.974	9.481
DRRN [13]		37.74	0.959	8.671	33.23	0.914	8.320	32.05	0.897	7.613	31.23	0.919	8.917	37.92	0.976	9.268
MS-LapSRN-D5R2 (ours)		37.62	0.960	9.038	33.13	0.913	8.539	31.93	0.897	7.776	30.82	0.915	9.081	37.38	0.975	9.434
MS-LapSRN-D5R5 (ours)		37.72	0.960	9.265	33.24	0.914	8.726	32.00	0.898	7.906	31.01	0.917	9.334	37.71	0.975	9.710
MS-LapSRN-D5R8 (ours)		37.78	0.960	9.305	33.28	0.915	8.748	32.05	0.898	7.927	31.15	0.919	9.406	37.78	0.976	9.765

Результаты

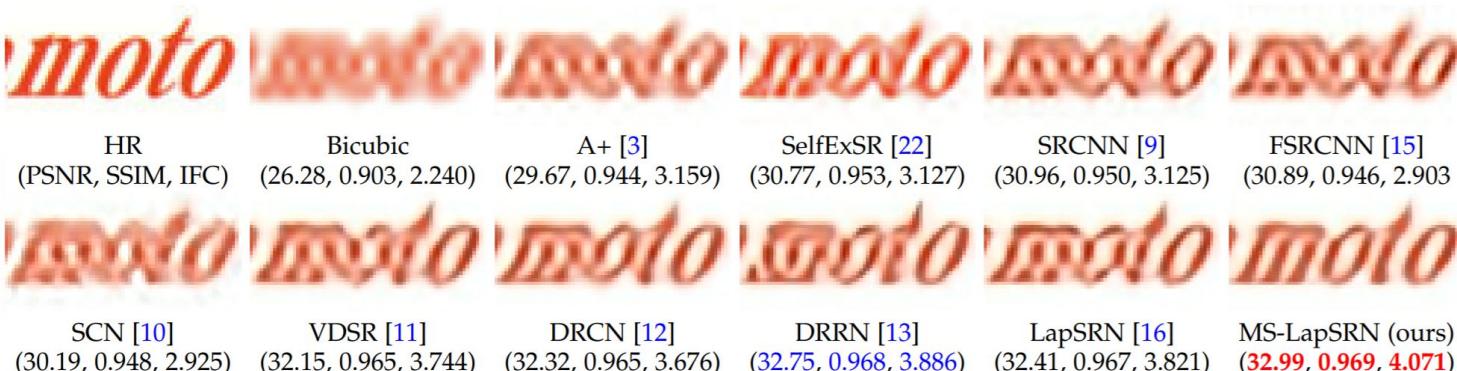
Algorithm	Scale	SET5			SET14			BSDS100			URBAN100			MANGA109		
		PSNR	SSIM	IFC	PSNR	SSIM	IFC	PSNR	SSIM	IFC	PSNR	SSIM	IFC	PSNR	SSIM	IFC
Bicubic		24.40	0.658	0.836	23.10	0.566	0.784	23.67	0.548	0.646	20.74	0.516	0.858	21.47	0.650	0.810
A+ [3]		25.53	0.693	1.077	23.89	0.595	0.983	24.21	0.569	0.797	21.37	0.546	1.092	22.39	0.681	1.056
RFL [5]		25.38	0.679	0.991	23.79	0.587	0.916	24.13	0.563	0.749	21.27	0.536	0.992	22.28	0.669	0.968
SelfExSR [22]		25.49	0.703	1.121	23.92	0.601	1.005	24.19	0.568	0.773	21.81	0.577	1.283	22.99	0.719	1.244
SRCNN [9]		25.33	0.690	0.938	23.76	0.591	0.865	24.13	0.566	0.705	21.29	0.544	0.947	22.46	0.695	1.013
FSRCNN [15]		25.60	0.697	1.016	24.00	0.599	0.942	24.31	0.572	0.767	21.45	0.550	0.995	22.72	0.692	1.009
SCN [10]	8×	25.59	0.706	1.063	24.02	0.603	0.967	24.30	0.573	0.777	21.52	0.560	1.074	22.68	0.701	1.073
VDSR [11]		25.93	0.724	1.199	24.26	0.614	1.067	24.49	0.583	0.859	21.70	0.571	1.199	23.16	0.725	1.263
DRCN [12]		25.93	0.723	1.192	24.25	0.614	1.057	24.49	0.582	0.854	21.71	0.571	1.197	23.20	0.724	1.257
LapSRN [16]		26.15	0.738	1.302	24.35	0.620	1.133	24.54	0.586	0.893	21.81	0.581	1.288	23.39	0.735	1.352
DRRN [13]		26.18	0.738	1.307	24.42	0.622	1.127	24.59	0.587	0.891	21.88	0.583	1.299	23.60	0.742	1.406
MS-LapSRN-D5R2 (ours)		26.20	0.747	1.366	24.45	0.626	1.170	24.61	0.590	0.920	21.95	0.592	1.364	23.70	0.751	1.470
MS-LapSRN-D5R5 (ours)		26.34	0.752	1.414	24.57	0.629	1.200	24.65	0.591	0.938	22.06	0.597	1.426	23.85	0.756	1.538
MS-LapSRN-D5R8 (ours)		26.34	0.753	1.435	24.57	0.629	1.209	24.65	0.592	0.943	22.06	0.598	1.446	23.90	0.759	1.564

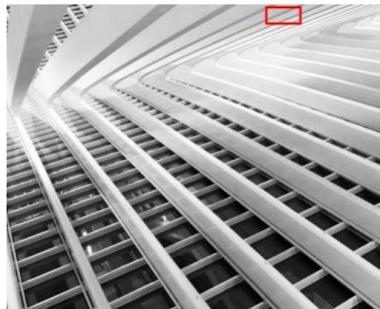


Ground-truth HR
URBAN100: img004



Ground-truth HR
MANGA109: MukoukizuNoCho





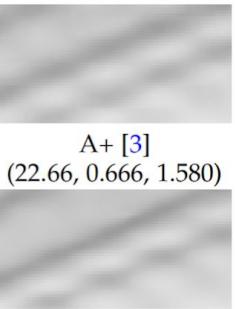
Ground-truth HR
URBAN100: img-042



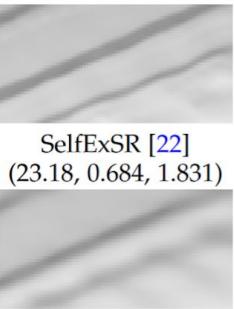
HR
(PSNR, SSIM, IFC)
(22.85, 0.676, 1.510)



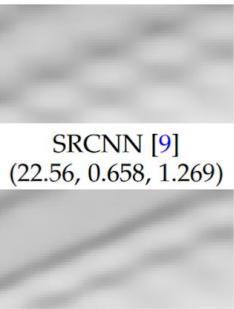
Bicubic
(21.67, 0.624, 1.160)



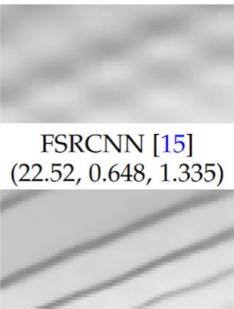
A+ [3]
(22.66, 0.666, 1.580)



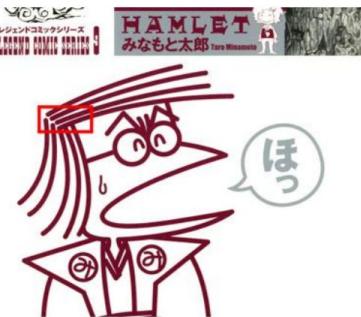
SelfExSR [22]
(23.18, 0.684, 1.831)



SRCNN [9]
(22.56, 0.658, 1.269)



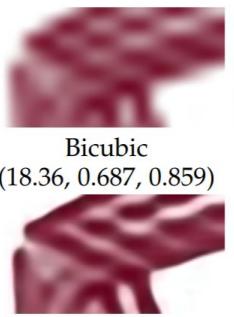
FSRCNN [15]
(22.52, 0.648, 1.335)



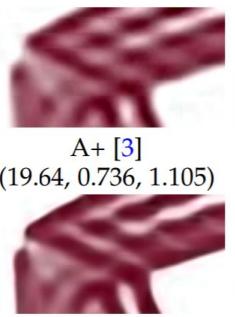
Ground-truth HR
MANGA109: Hamlet



HR
(PSNR, SSIM, IFC)
(18.36, 0.687, 0.859)



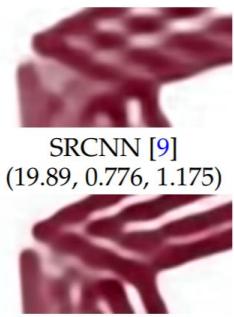
Bicubic
(18.36, 0.687, 0.859)



A+ [3]
(19.64, 0.736, 1.105)



SelfExSR [22]
(20.41, 0.789, 1.324)



SRCNN [9]
(19.89, 0.776, 1.175)



FSRCNN [15]
(19.83, 0.737, 1.028)

SCN [10]
(19.97, 0.770, 1.170)

VDSR [11]
(21.12, 0.828, 1.668)

DRCN [12]
(21.24, 0.827, 1.680)

DRRN [13]
(21.72, 0.848, 1.953)

LapSRN [16]
(21.36, 0.838, 1.731)

MS-LapSRN (ours)
(**21.94**, **0.862**, **2.124**)

Достоинства MS-LapSRN:

- 1) Может восстанавливать картинку в нескольких разрешениях одновременно
- 2) Время работы: работает быстрее большинства предыдущих моделей при лучшем качестве работы

Итоги

- 1) Были рассмотрены некоторые задачи компьютерного зрения, такие как:
 - сегментация
 - детекция
 - перенос стиля
 - повышение разрешения
- 2) Показаны методы их решения, основанные на глубинном обучении, в частности на CNN

Использованные источники

- R-CNN: <https://arxiv.org/pdf/1311.2524.pdf>
- Fast R-CNN: <https://arxiv.org/pdf/1504.08083.pdf>
- Faster R-CNN: <https://arxiv.org/pdf/1506.01497.pdf>
- YOLO: <https://arxiv.org/pdf/1506.02640v5.pdf>
- Mask R-CNN: <https://arxiv.org/pdf/1703.06870v3.pdf>
- Stanford, Detection and Segmentation (Lecture):
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf
- SRCNN: <https://arxiv.org/pdf/1501.00092.pdf>
- SRDenseNet:
https://openaccess.thecvf.com/content_ICCV_2017/papers/Tong_Image_Super-Resolution_Using_ICCV_2017_paper.pdf
- MS-LapSRN: <https://arxiv.org/pdf/1710.01992.pdf>
- Neural Style Transfer <https://arxiv.org/pdf/1508.06576.pdf>