Object Detection II: R-CNN, Fast R-CNN, Faster R-CNN

Антон Витвицкий

Otus, 2020

R-CNN: Rich feature hierarchies for accurate object detection and semantic segmentation, 2014

tags: *R-CNN, Object Detection, Selective Search*

R-CNN, Selective Search

Авторы предложили использовать *CNN* для задачи детекции, используя *bottom-up* стратегию *regions with CNN features*. Также показали эффективность *pretraining* и последующего *domain-specific finetuning*. Модель показала 30% прирост к *mAP* в сравнении с предыдущими SOTA; является относительно быстрой (на 2014) и хорошо масштабируемой. SOTA того времени являлись сложными ансамблями, комбинирующими низкоуровненые фичи с высокоуровневым контекстом из детекторов объектов и классификаторов сцен.

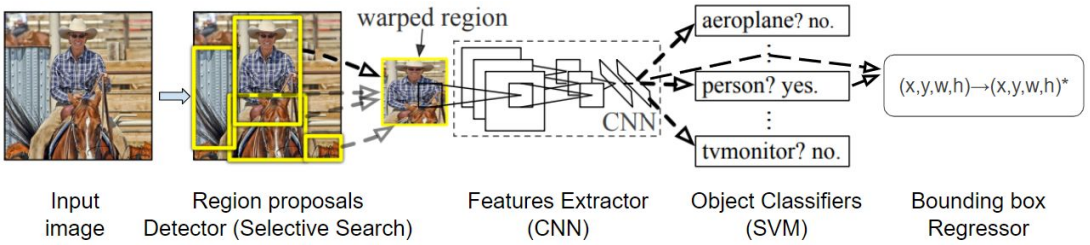


Рис. 1

2. Object detection with R-CNN

R-CNN состоит из четырех модулей (*рис.1*):

1) *Region proposals Detector*. Детектор не зависит от класса объекта и выдает *region proposals* (*RPs* / *регионы-кандидаты* - области изображения которые могут содержать объект).
2) *Features Extractor. CNN* извлекает фичи из каждого *RP*.
3) *Object Classifiers*. Набор линейных *SVM*-классификаторов, каждый из которых обучен распознавать конкретный класс объекта.
4) *Bounding box Regressor*. Уточняет координаты *RP* в зависимости от класса объекта.

2.1. Module design

В качестве *RP*-алгоритма используется *Selective Search (рис. 2)*, но можно использовать и другой.

В качестве *Feature Extractor* используется *AlexNet* (слои: 5 conv и 2 fully connected). На вход поступает rgb 227×227, на выходе - вектор фич размером 4096.

Так как *AlexNet* не *FCN* (*Fully Connected Network*), то изображение для каждого *RP* необходимо привести к фиксированному размеру входа сети (227x227). На *рис. 3* представлены варианты такой трансформации:

A) исходный *RP*.
B) *tightest square with context* (плотный квадрат с контекстом). *RP* вписывается в квадрат и скейлится до нужного размера (изотропно, т.е. с сохранением пропорций).
C) *tightest square without context* (плотный квадрат без контекста). *RP* дополняется до квадратной формы нулями (*padding*) и затем скейлится до нужного размера.
D) *warp* (деформация). *RP* скейлится до нужного размера без сохранения пропорций.

Для каждого варианта также рассматривается трансформация *без контекста* и *с контекстом* (верхняя и нижняя строка на *рис.3*, соответственно). Для трансформации *с контекстом* сперва применяется *dilate* так, что итоговое изображение (227×227) содержит ровно *p=16* пикселей, которые принадлежат области вокруг исходного *RP*. В итоге, авторы остановились на варианте (D) *warp с контекстом*. Такой способ выбран за простоту.

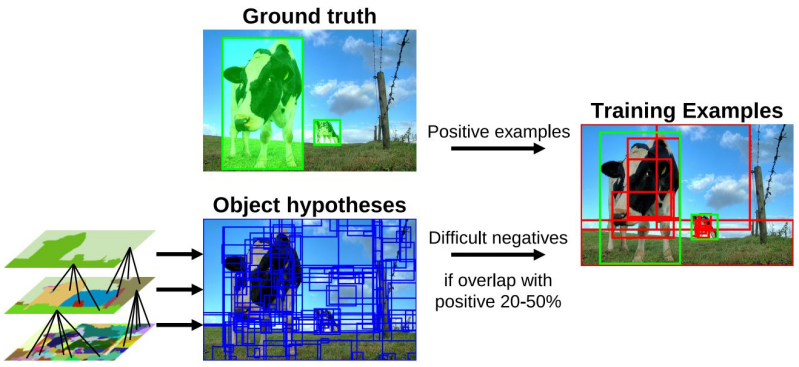


Рис. 2

Рис. 3

2.2. Test-time detection

Для каждого изображения во время тестирования:

1) При помощи *Selective Search ("fast mode")* извлекаются 2000 *RP*s.
2) Каждый *RP* трансформируется, прогоняется через CNN, и извлекаются фичи 2000×4096.
3) Каждый *SVM* (обученный на конкретный класс) классифицирует каждую из 2000 фич.
4) Для каждого класса независимо применяется *greedy Non-Maximum Suppression (NMS)*, который отбрасывает регион, если тот имеет *IoU* больше заданного порога с регионом с более высокий *confidence score* (для того же класса).
5) Для каждого *RP* который классифицирован как объект, применяется соответствующий (его классу) *bounding box* регрессор, который уточняет его координаты.

Run-time analysis. Метод работает (относительно) быстро, т.к.: 1) CNN не зависит от класса объекта и не использует *sliding window*; 2) не используется *spatial pyramid*; 3) низкая размерность вектора фич в сравнении с другими методами (на 1-2 порядка). Число параметров: CNN + 2000×4096 (фичи) + 4096×*N* (веса *SVM*, где *N* - число классов).

Scale invariant. **(?)** Чем можно достичь этого в *object detection*: 1) *spatial pyramid*; 2) *the brute-force approach* - изображения фиксированного размера, сеть сама обучается быть инвариантной.

2.3. Training

Supervised pre-training. CNN предобучена на *ILSVRC2012* (только метки классов, боксов нет).

Domain-specific fine-tuning. Далее CNN дотюнивается на warped *RPs* (последний слой 1000-way заменяется на слой (*N*+1)-way, где *N* - число классов (*N*=20 для *VOC*, и *N*=200 для *ILSVRC2013*). Каждый *PR* имеющий ≥ *0.5 IoU* с *ground-truth* боксом считается положительным примером для класса, остальные - негативными. Размер батча 128, 32 - положительных (любого класса), 96 - фон.

Object category classifiers. На каждый класс обучается один SVM. Положительными примерами считаются только *gt* боксы; боксы имеющие *<0.3 IoU* со всеми *gt* для этого класса считаются негативными, все остальные - игнорируются. Используется *hard negative mining* стратегия. Позже обнаружили что SVM можно заменить на *softmax* ^_^.

Bounding box regression. После SVM идет регрессор (свой для каждого класса), который уточняет боксы. Идея в том, что регрессор обучается *трансформировать RP*-боксы в *gt*-боксы: на вход алгоритму обучения поступают пары {(***P****i*,***G****i*)}, где ***P****i=(Px,Py,Pw,Ph)* - центр и ширина/высота (в пикселях) для *RP* ***P****i*, ***G****i=(Gx,Gy,Gw,Gh)* - то же для *gt* ***G****i*.

Трансформация параметризуется функциями ***D****x(P)*, ***D****y(P)*, ***D****w(P)*, ***D****h(P)* - *scale-invariant translation* центра бокса *P* и *log-space translation* его ширины/высоты: ***D*****(P)=****w*****·****φ****5(P)*, где ***φ****5(P)* - фичи из *max_pool5* слоя, ***w**** - вектор весов (транспонированный). Веса обучаются оптимизацией the *regularized least squares objective (ridge regression)*:

$$\mathbf{w}_\star = \underset{\hat{\mathbf{w}}_\star}{\operatorname{argmin}} \sum_i^N (t_\star^i - \hat{\mathbf{w}}_\star^{\mathrm{T}} \phi_5(P^i))^2 + \lambda \|\hat{\mathbf{w}}_\star\|^2 .$$

где ***t**** для пары *(P,G)*: ***t**x=(Gx-Px)/Pw*, ***t**y=(Gy-Py)/Ph*, ***t**w=log(Gw/Pw)*, ***t**h = log(Gh/Ph)*.

После обучения ***D***(P)* функций можно трансформировать предсказанный *RP* ***P**i* в *gt* ***G**i^:*

***G**x^=Pw · Dx(P)+Px*, ***G**y^=Ph · Dy(P)+Py*, ***G**w^=Pw · exp(Dw(P))*, ***G**h^=Ph · exp(Dh(P))*.

Вывод формул: ***G**x^=Pw · Dx(P)+Px* **=** *Pw · (Gx-Px)/Pw+Px* **=** ***G**x.*

***G**w^=Pw · exp(Dw(P))* **=>** *log(**G**w^)=log(Pw · exp(Dw(P)))* **=** *log(Pw)+Dw(P)* **=** *log(Pw) · log(Gw/Pw)* **=** *log(Pw)+log(Gw)+log(1/Pw)* **=** *log(Gw)* **=>** ***G**w^* **=** *Gw.*

3.1. Visualizing learned features

Авторы демонстрируют чему их сеть обучается. Для этого они берут *max pool* 6×6×256, идущий после последнего *conv* слоя и имеющий *receptive field* 195×195 (на картинке 227×227). Далее получают 10m *RPs* и сортируют результат по силе активации конкретных нейронов в *max pool* (*рис. 4*). Оказывается, что отдельные нейроны обучились активироваться на конкретные фичи (лица, собаки, наборы точек, текст, красные пятна и т.д.). Далее *fc* слой строит композиции этих фич.

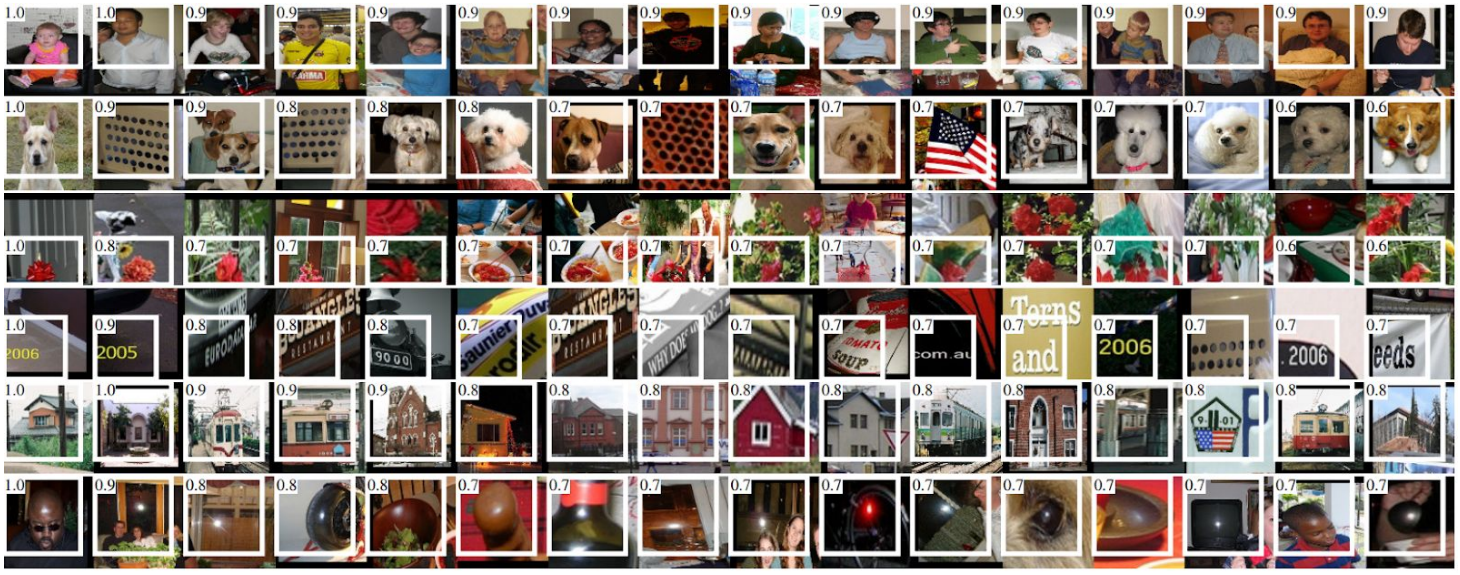


Рис.4

3.2 Results

На *рис.5* представлено сравнение результатов R-CNN с SOTA методами (2014) на VOC 2010. R-CNN показал прирост к *mAP* на 25% по сравнениею с предыдущим лучшим результатом *SegDPM* модели (*deformable part model* - графовая модель, sliding window подход и HOG фичи). Методы *UVA* и *Regionlets* также используют *Selective Search*.

VOC 2010 test	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
DPM v5 [20]†	49.2	53.8	13.1	15.3	35.5	53.4	49.7	27.0	17.2	28.8	14.7	17.8	46.4	51.2	47.7	10.8	34.2	20.7	43.8	38.3	33.4
UVA [39]	56.2	42.4	15.3	12.6	21.8	49.3	36.8	46.1	12.9	32.1	30.0	36.5	43.5	52.9	32.9	15.3	41.1	31.8	47.0	44.8	35.1
Regionlets [41]	65.0	48.9	25.9	24.6	24.5	56.1	54.5	51.2	17.0	28.9	30.2	35.8	40.2	55.7	43.5	14.3	43.9	32.6	54.0	45.9	39.7
SegDPM [18]†	61.4	53.4	25.6	25.2	35.5	51.7	50.6	50.8	19.3	33.8	26.8	40.4	48.3	54.4	47.1	14.8	38.7	35.0	52.8	43.1	40.4
R-CNN	67.1	64.1	46.7	32.0	30.5	56.4	57.2	65.9	27.0	47.3	40.9	66.6	57.8	65.9	53.6	26.7	56.5	38.1	52.8	50.2	50.2
R-CNN BB	**71.8**	**65.8**	**53.0**	**36.8**	**35.9**	**59.7**	**60.0**	**69.9**	**27.9**	**50.6**	**41.4**	**70.0**	**62.0**	**69.0**	**58.1**	**29.5**	**59.4**	**39.3**	**61.2**	**52.4**	**53.7**

Рис.5

Fast R-CNN, 2015

tags: *R-CNN, Object Detection, VGG, Region proposals, RoI pooling*

paper, link2

Метод называется *Fast Region-based Convolutional Network*. Построен на основе *R-CNN* с использованием нескольких инноваций, что позволило повысить точность детекции, а также повысить скорость работы модели: в фазе обучения быстрее *R-CNN* в 9 раз, а в фазе инференса в 213 раз, или 0.3 сек на картинку на K40 (без *object proposal* части). Для экстракции фич используется *VGG16*.

1.1 R-CNN drawbacks (?)

- *Обучение происходит в несколько этапов*. Сперва дотюнивается *VGG16 (*экстрактор фич, в оригинале *AlexNet)*, потом на фичах из *VGG16* обучаются *SVMs* как *object classifiers*, и наконец учится *bounding box regressor (BBR)*.
- *Обучение слишком затратно* по времени, и по памяти. Из каждого изображения в датасете извлекается 2000x4096=32MB фич для обучения *SVM* и *BBR*, что требует сотен GB данных.
- *Слишком медленная детекция*. Вывод модели занимает 47 секунд на одно изображение.

1.2 Fast R-CNN advantages

- Более высокая оценка *mAP*, чем у R-CNN.
- Модель расшаривает веса и обучается *end-to-end*, используя *multi-task loss*.
- Не требует дополнительного места на диске для кэширования фич.

2. Fast R-CNN architecture

На рис. 1 представлена архитектура *Fast R-CNN*:

1) На вход подается изображение и набор *region proposals* (*RPs*, выход *Selective Search*).
2) Из изображения извлекается *feature map* при помощи *VGG16*.
3) Для каждого *RP* при помощи *region of interest (RoI) pooling layer* извлекается *вектор фич* фиксированного размера из *feature map*.
4) Далее каждый *вектор фич* подается на вход последовательности из *fully connected* (*FC*) слоев, которая в конце разветвляется на два выходных слоя: a) классификатор объектов, который имеет *K+1* выходов (*K*-классов + фон) с навешенным *softmax*; b) *BBR*, который имеет *4xK* выходов (4 значения для каждого класса уточняющие *bounding box* объекта).

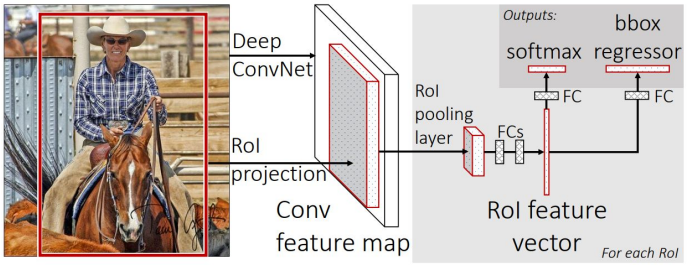


Рис. 1

2.1. The RoI pooling layer

Использует *max pooling* чтобы превратить фичи внутри любого *RoI* в небольшую *feature map* фиксированного размера *H×W* (например, 7×7). Здесь *RoI* - прямоугольное окно в *feature map* из *CNN*, где координаты окна берутся из соответствующего *RP*. Предположим, у нас есть *RoI* произвольного размера h×w (*рис.2,* слева), и мы хотим преобразовать его в *output feature map* фиксированного размера *H×W* (*рис.2,* справа) при помощи *max pooling.* Тогда область для каждой *pooling area* будет равна *h/H×w/W* (*рис.2,* по середине). Например, *RoI* = 5×7, а *output feature map* =

2×2, тогда *pooling area* будет = 2×2 или 3×3 после округления. Таким образом, максимальное значение в каждой клетке *output feature map* будет максимумом из соответствующей области *RoI*.

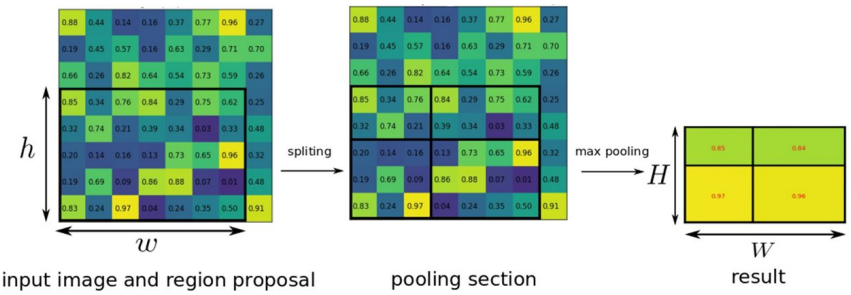


Рис. 2

2.2 Training

Сперва *VGG16* предобучается на 1000 классах *ImageNet*, далее дотюнивается на *PASCAL VOC* или *MS COCO*. Обучение end-to-end, *softmax* и *BBR* учатся совместно. В качестве батча берется *N* изображений и *R/N RoI* для них. Авторы берут *N*=2 и *R*=128, т.е. по 64 *RoI* для каждого изображения. Такая схема принята для ускорения обучения (т.к. для всех 64 *RoI* нужно лишь один раз прогнать изображение через сеть). В батче 25% *RoI* положительные (содержащие какой-то объект), остальные - негативные. Положительным считается *RoI*, если его *RP* имеет *IoU>=0.5* с *gt* боксом. Негативными - если *IoU* в диапазоне [0.1;0.5). RoI с *IoU*<0.1 отбрасываются (*hard negative mining*).

2.3 Multi-task loss

Для каждого *RoI* в обучении есть *gt*: *u* - класс объекта, *v*=(v_x, v_y, v_w, v_h) - *bounding box* (параметризуется аналогично R-CNN, т.е. *scale-invariant translation* центра бокса (*vx,vy)* и *log-space translation* ширины/высоты (*vw,vh*) относительно *RP*).

Весь лосс: $L(p, u, t^u, v) = L_{\text{cls}}(p, u) + \lambda[u \geq 1]L_{\text{loc}}(t^u, v)$, где $L_{\text{cls}}(p, u) = -\log p_u$ это *loss log* для true класса *u; p* - предсказанный класс объекта; *tu* - предсказанный бокс; L_{loc} - лосс для *BBR*; $[u \geq 1]$ указывает что лосс применяется только к объектам, т.е. равно *1* для объектов, и *u=0* для фона.

Лосс для *BBR*: $L_{\text{loc}}(t^u, v) = \sum_{i \in \{x,y,w,h\}} \text{smooth}_{L_1}(t_i^u - v_i)$, где $\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$.

Коэффициент λ - отвечает за баланс между лоссами, во всех экспериментах авторов λ=1.

2.4 Fast R-CNN detection

Во время теста на вход подается изображение и 2000 *RPs* из *Selective Search* (экспериментировали с большим числом *RPs*, но результаты ухудшались). На выходе получаем *RoI*, которые были распознаны как объект, и для всех *RoI* каждого класса применяется *non-maximum suppression* (аналогично R-CNN), после которого смежные *RoI* одного класса сливаются в один.

Для достижения *scale invariant* авторы используют две стратегии: a) *brute-force*, когда модель явно обучается быть *scale invariant* из данных; b) *image pyramid* - изображения разных размеров.

3. Main results

Оценивалось три варианта *backbone CNN*: 1) **S**=*AlexNet* 2) **M**=*VGG-like* версия **S** 3) **L**=*VGG16*.

На *puc.3* представлены результаты сравнения разных *backbone* и вариантов обучения (*stage-wise* вариант - *softmax* и *BBR* обучаются раздельно).

	S				M				L			
multi-task training?		✓		✓		✓		✓		✓		✓
stage-wise training?			✓				✓				✓	
test-time bbox reg?			✓	✓			✓	✓			✓	✓
VOC07 mAP	52.2	53.3	54.6	**57.1**	54.7	55.5	56.6	**59.2**	62.6	63.4	64.0	**66.9**

	SPPnet ZF		S		M		L
scales	1	5	1	5	1	5	1
test rate (s/im)	0.14	0.38	**0.10**	0.39	0.15	0.64	0.32
VOC07 mAP	58.0	59.2	57.1	58.4	59.2	60.7	**66.9**

Рис.3 *Рис.4*

3.1 Multi Scale Training and Testing

На *рис.4* представлены результаты теста на 1 масштабе изображения и на 5 (*image pyramid*). Видно что *image pyramid* дает прирост к *mAP*, однако просаживает скорость работы модели.

3.2 SVM vs Softmax

На *рис.5* показано, что *Fast R-CNN* (*FRCN*) с *softmax* дает лучшую точность. Кроме того, *softmax* не требует хранить промежуточные данные на диске, и позволяет обучать модель end-to-end.

	method	classifier	S	M	L
SVM	R-CNN [9, 10]	SVM	**58.5**	**60.2**	66.0
	FRCN [ours]	SVM	56.3	58.7	66.8
softmax	FRCN [ours]	softmax	57.1	59.2	**66.9**

Рис. 5

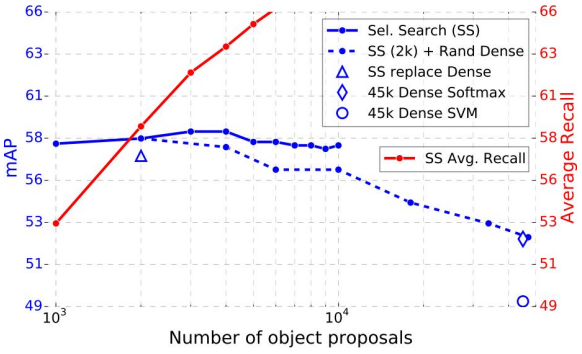


Рис. 8

3.3 PASCAL VOC 2010/2012

На *рис.6* и *рис.7* показаны результаты на P*ASCAL VOC 2010* и *2012*, соответственно.

method	train set	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	persn	plant	sheep	sofa	train	tv	mAP
BabyLearning	Prop.	77.7	73.8	62.3	48.8	45.4	67.3	67.0	80.3	41.3	70.8	49.7	79.5	74.7	78.6	64.5	36.0	69.9	55.7	70.4	61.7	63.8
R-CNN BB [10]	12	79.3	72.4	63.1	44.0	44.4	64.6	66.3	84.9	38.8	67.3	48.4	82.3	75.0	76.7	65.7	35.8	66.2	54.8	69.1	58.8	62.9
SegDeepM	12+seg	**82.3**	75.2	67.1	50.7	**49.8**	71.1	69.6	88.2	42.5	71.2	50.0	85.7	76.6	81.8	69.3	**41.5**	**71.9**	62.2	73.2	**64.6**	67.2
FRCN [ours]	12	80.1	74.4	67.7	49.4	41.4	74.2	68.8	87.8	41.9	70.1	50.2	86.1	77.3	81.1	70.4	33.3	67.0	63.3	77.2	60.0	66.1
FRCN [ours]	07++12	82.0	**77.8**	**71.6**	**55.3**	42.4	**77.3**	**71.7**	**89.3**	**44.5**	**72.1**	**53.7**	**87.7**	**80.0**	**82.5**	**72.7**	36.6	68.7	**65.4**	**81.1**	62.7	**68.8**

Рис. 6

method	train set	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	persn	plant	sheep	sofa	train	tv	mAP
BabyLearning	Prop.	78.0	74.2	61.3	45.7	42.7	68.2	66.8	80.2	40.6	70.0	49.8	79.0	74.5	77.9	64.0	35.3	67.9	55.7	68.7	62.6	63.2
NUS_NIN_c2000	Unk.	80.2	73.8	61.9	43.7	**43.0**	70.3	67.6	80.7	41.9	69.7	51.7	78.2	75.2	76.9	65.1	**38.6**	**68.3**	58.0	68.7	63.3	63.8
R-CNN BB [10]	12	79.6	72.7	61.9	41.2	41.9	65.9	66.4	84.6	38.5	67.2	46.7	82.0	74.8	76.0	65.2	35.6	65.4	54.2	67.4	60.3	62.4
FRCN [ours]	12	80.3	74.7	66.9	46.9	37.7	73.9	68.6	87.7	41.7	71.1	51.1	86.0	77.8	79.8	69.8	32.1	65.5	63.8	76.4	61.7	65.7
FRCN [ours]	07++12	**82.3**	**78.4**	**70.8**	**52.3**	38.7	**77.8**	**71.6**	**89.3**	**44.2**	**73.0**	**55.0**	**87.5**	**80.5**	**80.8**	**72.0**	35.1	**68.3**	**65.7**	**80.4**	**64.2**	**68.4**

Рис. 7

3.4 Region Proposals

Авторы обнаружили, что повышение числа *RPs* не приводит к повышению *mAP* (см. *рис.8*). Слабым местом *Fast R-CNN* остается необходимость в использовании стороннего *Region Proposals*.

Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, 2016

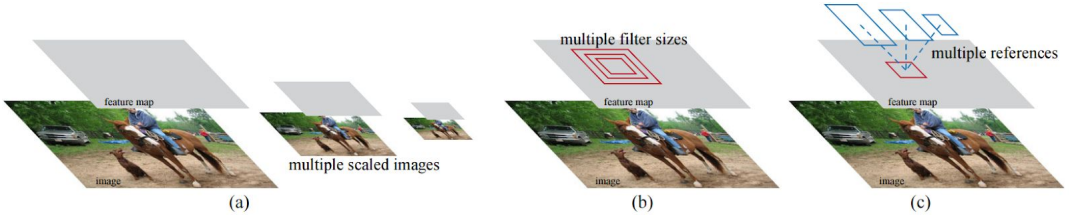
tags: *R-CNN, Object Detection, RPN*

paper

Авторы замечают, что в *Object Detection* алгоритмах (*SPPnet, Fast R-CNN*) узким местом является вычисление *region proposals (RP)*. Поэтому они берут *Fast R-CNN* модель и заменяют *RP*-алгоритм (*Selective Search)* на *Region Proposals Network* (*RPN*), которая является *Fully Convolution Network* (*FCN)* и расшаривает веса с *detection* сетью (*VGG16*). В итоге *Faster R-CNN* работает как единая сеть, в которой *RPN* выступает как *attention* и указывает сети где объект искать. Авторы получили state-of-the-art результаты на PASCAL VOC 2007/2012 и MS COCO 2015. Кроме того, *Faster R-CNN* может работать в реалтайме на скорости 5 кадров в секунду (FPS) на K40 GPU, а также успешно применятся к другим задачам: *instance segmentation*, *3d object detection*, *image captioning* и др.

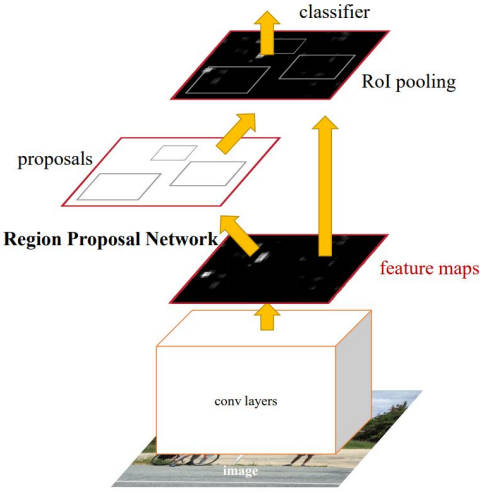
1. FASTER R-CNN

В предыдущих работах, чтобы сделать модель инвариантной к масштабу входного изображения (*scale invariant)* применялись следующие подходы: a) *images pyramid* (*puc.1a*), в котором *feature maps* вычислялись на изображении в разных масштабах; b) *filters pyramid* (*puc.2b*), в котором к *feature map* применялись фильтры разных масштабов. В Faster R-CNN же используется *pyramids of reference boxes* (*puc.1c*), в котором из *feature map* извлекаются боксы разных масштабов.

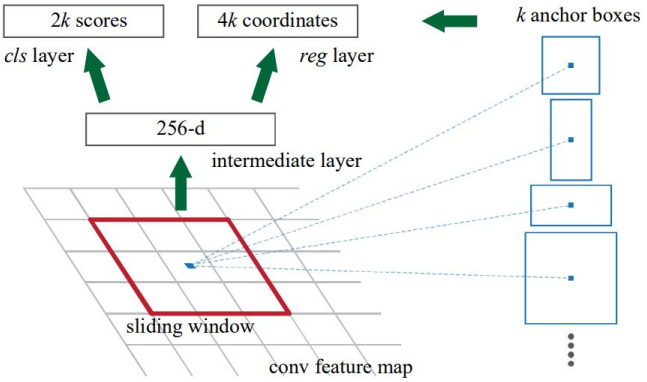


Puc. 1

Faster R-CNN это единая сеть, состоящая из двух модулей: *RPN* и *Fast R-CNN* детектор (*puc.2*).



Puc.2



Puc.3

1.1 Region Proposal Networks (RPN)

RPN принимает на вход изображение произвольного размера (т.к. *FCN*) и выдает список *RPs* с *objectness score* (вероятность что это объект). Обе сети *RPN* и *Fast R-CNN* используют общую сеть (*VGG16*) как экстрактор фич. *RPN* скользит окном *n×n* по *feature map* последнего *conv* слоя *VGG16* (*puc.3*) и извлекает вектор фич размера *512*. Далее эти фичи поступают на вход двух *полносвязных* слоев: *box-regression* (*reg*) и *box-classification* (*cls*). Авторы используют *n=3*, что дает большой *receptive field* равный *228* на сети *VGG16* (размер входа - 600 пикселей меньшая сторона). Такая архитектура реализована как *conv 3×3* с последующим *conv 1×1* для *reg* и *conv 1×1* для *cls*.

1.1.1 Anchors

Для каждой позиции скользящего окна *RPN* на *feature map* одновременно предсказывается несколько *RPs*, где максимальное число *RPs* для каждой позиции обозначается *k*. Таким образом, *reg* слой имеет выход *4k*, кодирующий координаты *k* боксов, и *cls* слой имеет выход *2k* - вероятности что это объект или нет для *k* proposals (реализован как *2-way softmax*). Эти *k RPs* параметризуются *k* референсными боксами, которые будем называть *anchors* (якорные токи). Каждый *anchor* отцентрирован в позиции скользящего окна и ассоциирован с некоторым масштабом и отношением сторон. Авторы используют три масштаба и три отношения сторон, получая в итоге *k=9 anchors* для каждой позиции окна. Для *feature map H×W* существует в целом *HWk anchors*. Такой подход делает *RPN* инвариантной к сдвигам и масштабу, и убирает необходимость в *images* / *features pyramid*.

1.1.2 Loss Function

Во время обучения *RPN* каждому *anchors* присваивается бинарная метка. *Anchor* считается позитивным, если он имеет *IoU>0.7* с одним из *ground truth* (*gt*) боксом, и считается негативным, если *IoU<0.3* со всеми *gt* боксами. Остальные *anchors* не вносят вклад в обучение. Таким образом, один *gt* бокс может иметь несколько позитивных *anchors*. *И*спользуется *multi-task loss*:

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*).$$

где *i* - индекс *anchor* в mini-batch; *pi* - предсказанная вероятность, что *anchor* содержит объект; *p** - равен *1* для позитивных anchors и *0* для негативных; *ti* - вектор представляющий *4* параметризованные координаты предсказанного бокса; *t** - *gt* бокс ассоциированный с этим *anchor*.

Lcls использует *log-loss* на два класса (*softmax*), а $L_{reg}(t_i, t_i^*) = R(t_i - t_i^*)$, где *R - smooth L1 loss*: $\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$, множитель *p* Lreg* указывает, что loss только для позитивных *anchors*.

Оба loss нормализуются на *Ncls=256* (размер батча) и *Nreg~2400* (число *anchors*), соответственно. Балансировочный коэффициент *λ=10*, что делает вклад обоих loss равным.

1.1.3 Bounding Box Regression

Координаты для регрессии боксов параметризуются аналогично *R-CNN*:

$$\begin{aligned} t_{\text{x}} &= (x - x_{\text{a}})/w_{\text{a}}, \quad t_{\text{y}} = (y - y_{\text{a}})/h_{\text{a}}, \\ t_{\text{w}} &= \log(w/w_{\text{a}}), \quad t_{\text{h}} = \log(h/h_{\text{a}}), \\ t^*_{\text{x}} &= (x^* - x_{\text{a}})/w_{\text{a}}, \quad t^*_{\text{y}} = (y^* - y_{\text{a}})/h_{\text{a}}, \\ t^*_{\text{w}} &= \log(w^*/w_{\text{a}}), \quad t^*_{\text{h}} = \log(h^*/h_{\text{a}}), \end{aligned}$$

где *x,y,w,h* - центр бокса и его ширина/высота; *x, xa, x** - предсказанный бокс, *anchor* бокс и *gt* бокс, соответственно (аналогично для *y,w,h*). В такой постановке задачу можно рассматривать как регрессию *anchor* бокса к ближайшему *gt* боксу. Всего существует *k* регрессоров, которые не расшаривают веса, и каждый из которых ответственен за свой масштаб и соотношение сторон.

1.1.4 Training RPNs

RPN обучается end-to-end. Каждый mini-batch состоит из одного изображения и выборки из 256 *anchors*, с отношением позитивных *anchors* к негативным *1*:*1*. Сеть предобучается на *ImageNet*.

1.2 Sharing Features for RPN and Fast R-CNN

Обучается единая сеть, состоящая из *RPN* и *Fast R-CNN* с расшаренными conv слоями. Обучение происходит в четыре этапа следующим образом:

1) *RPN*, предобученная на *ImageNet*, обучается на *region proposals* задаче.

2) Детектор *Fast R-CNN*, предобученный на *ImageNet*, обучается на *proposals* из *RPN* шага 1. К этому моменту *RPN* и *Fast R-CNN* не расшаривают веса.
3) *RPN* инициализируется из *Fast R-CNN* сети после шага 2. Далее, все расшариваемые веса замораживаются, за исключением уникальных для *RPN* слоев, которые затем дотюниваются. К этому моменту обе сети расшаривают веса.
4) Аналогично для *Fast R-CNN*, все расшариваемые веса замораживаются, за исключением уникальных для *Fast R-CNN* слоев, которые затем дотюниваются.

1.3 Implementation Details

Изображение масштабируется так, что меньшая сторона становится равной *600* пикселям (*receptive field* для *anchors* равен *228*). Для *anchors* используется три масштаба (128^2, 256^2 и 512^2) и три отношения сторон (1:1, 1:2, 2:1). На *рис.4* представлен пример того, как *Faster R-CNN* способен находить объекты разных масштабов и пропорций. На *рис.5* показан результат экспериментов с различным числом масштабов и отношений сторон. На *рис.6* показан средний размер *proposals* для *anchors* каждого типа (получены в результате обучения модели).

Для типичного изображение размером *1000×600* модель выдает около *60×40×9=20000 anchors*. Хотя во время обучения граничные *anchors* отфильтровывалось (оставалось около *6000* для изображения *1000×600*), т.к. они ухудшают сходимость сети, модель по прежнему способна находить объекты на границах во время теста.

RPN выдает много пересекающихся *proposals*. Поэтому, чтобы избавиться от избыточности, применяется *non-maximum suppression* (*NMS*) на *proposals*, учитывая их оценку *cls*. В качестве порога берется *IoU>0.7*, после чего остается около *2000 proposals*. *NMS* не влияет на точность детекции, однако значительно снижает число *proposals*.

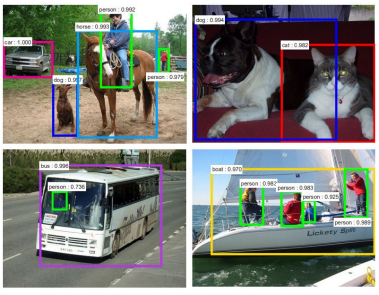


Рис.4

settings	anchor scales	aspect ratios	mAP (%)
1 scale, 1 ratio	128^2	1:1	65.8
	256^2	1:1	66.7
1 scale, 3 ratios	128^2	{2:1, 1:1, 1:2}	68.8
	256^2	{2:1, 1:1, 1:2}	67.9
3 scales, 1 ratio	$\{128^2, 256^2, 512^2\}$	1:1	**69.8**
3 scales, 3 ratios	$\{128^2, 256^2, 512^2\}$	{2:1, 1:1, 1:2}	**69.9**

Рис.5

anchor	128^2, 2:1	128^2, 1:1	128^2, 1:2	256^2, 2:1	256^2, 1:1	256^2, 1:2	512^2, 2:1	512^2, 1:1	512^2, 1:2
proposal	188×111	113×114	70×92	416×229	261×284	174×332	768×437	499×501	355×715

Рис.6

2. Main Results

На *рис.7* представлено сравнение результатов *Fast R-CNN* и *Faster R-CNN* на *MS COCO*.

method	proposals	training data	COCO val		COCO test-dev	
			mAP@.5	mAP@[.5, .95]	mAP@.5	mAP@[.5, .95]
Fast R-CNN [2]	SS, 2000	COCO train	-	-	35.9	19.7
Fast R-CNN [impl. in this paper]	SS, 2000	COCO train	38.6	18.9	39.3	19.3
Faster R-CNN	RPN, 300	COCO train	41.5	21.2	42.1	21.5
Faster R-CNN	RPN, 300	COCO trainval	-	-	**42.7**	**21.9**

Рис.7