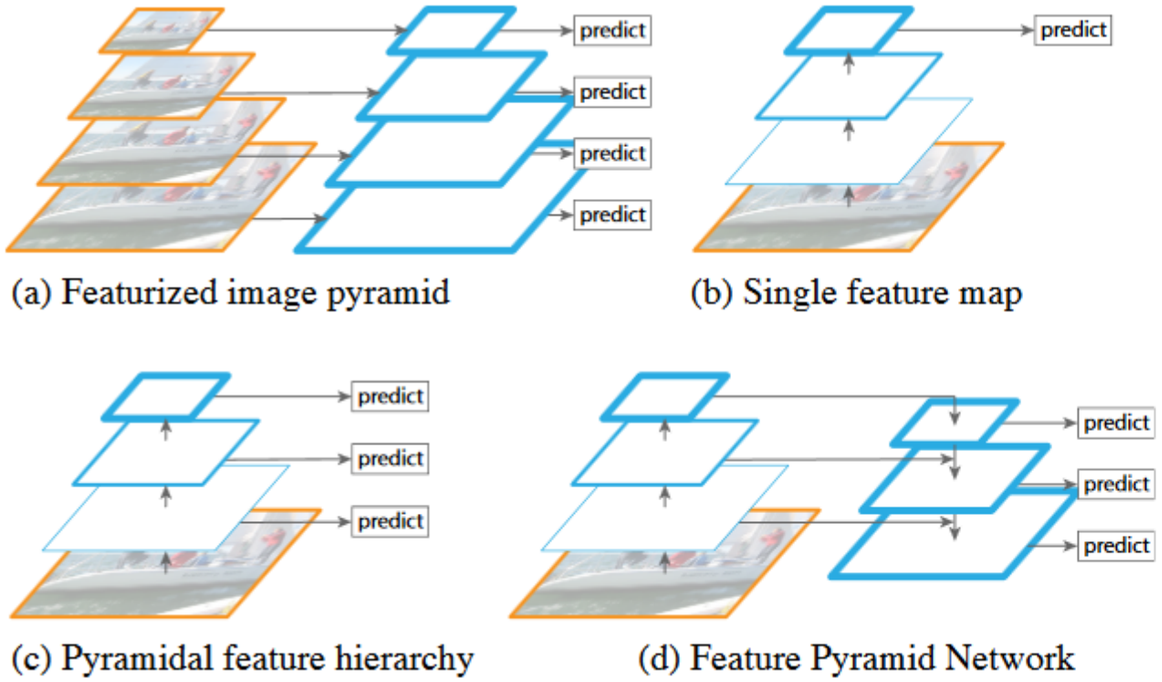
курс «Глубокое обучение»

Детектирование объектов на изображениях
лекция 2

Александр Дьяконов

Детектирование объектов: Feature Pyramid Networks (FPN)



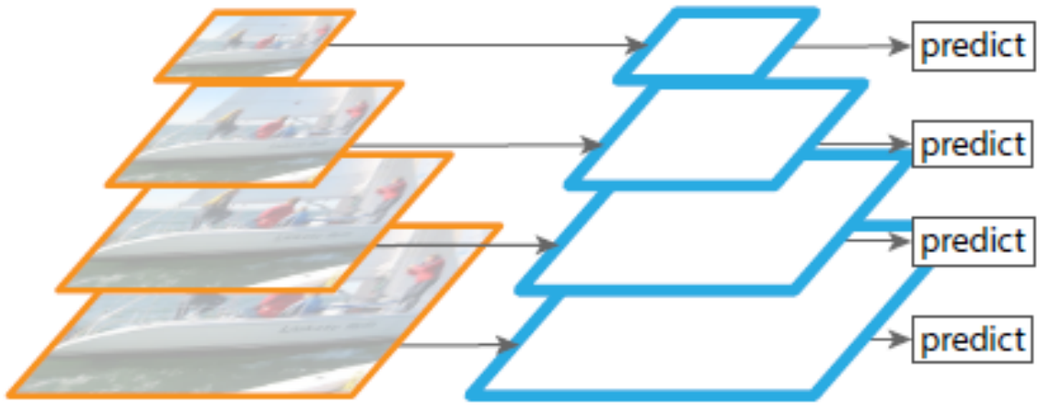
(a) Featurized image pyramid (b) Single feature map

(c) Pyramidal feature hierarchy (d) Feature Pyramid Network

Figure 1. (a) Using an image pyramid to build a feature pyramid. Features are computed on each of the image scales independently, which is slow. (b) Recent detection systems have opted to use only single scale features for faster detection. (c) An alternative is to reuse the pyramidal feature hierarchy computed by a ConvNet as if it were a featurized image pyramid. (d) Our proposed Feature Pyramid Network (FPN) is fast like (b) and (c), but more accurate. In this figure, feature maps are indicate by blue outlines and thicker outlines denote semantically stronger features.

для инвариантности к масштабу делают «пирамиды изображений»
обычно используются только во время тестирования (т.к. долгое end2end-обучение)
но свёрточные сети получают «пирамиды признаков»
это всё – способы получения карт признаков. Можно использовать с любыми архитектурами НС! https://arxiv.org/abs/1612.03144

Разные архитектуры

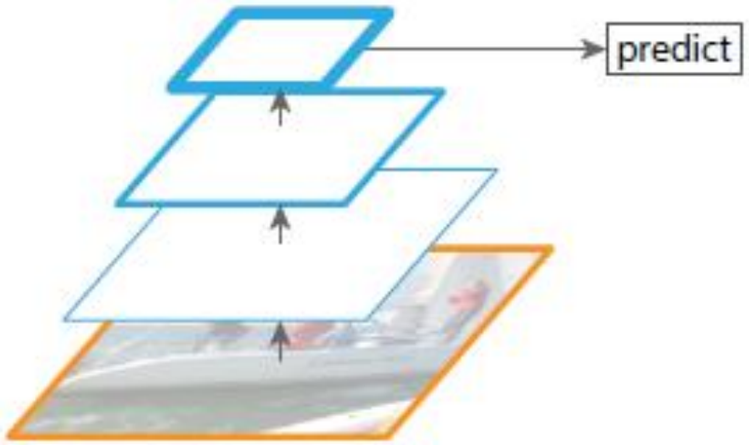


(a) Featurized image pyramid

Использовались ещё до эры DL

Точные

Медленные



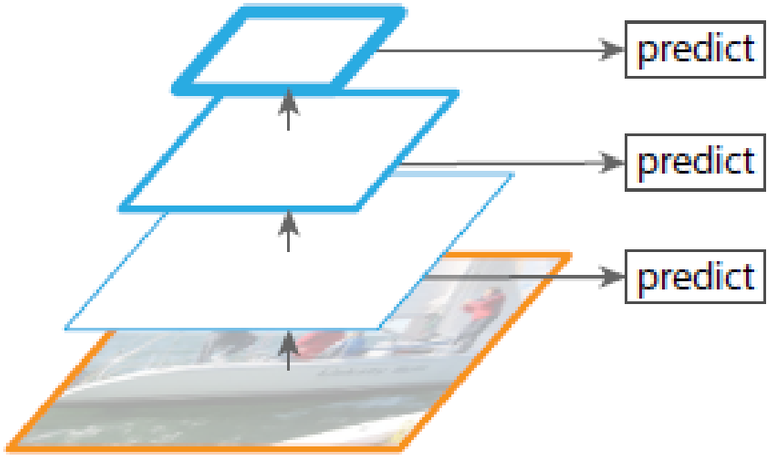
(b) Single feature map

Стандартное CNN-решение

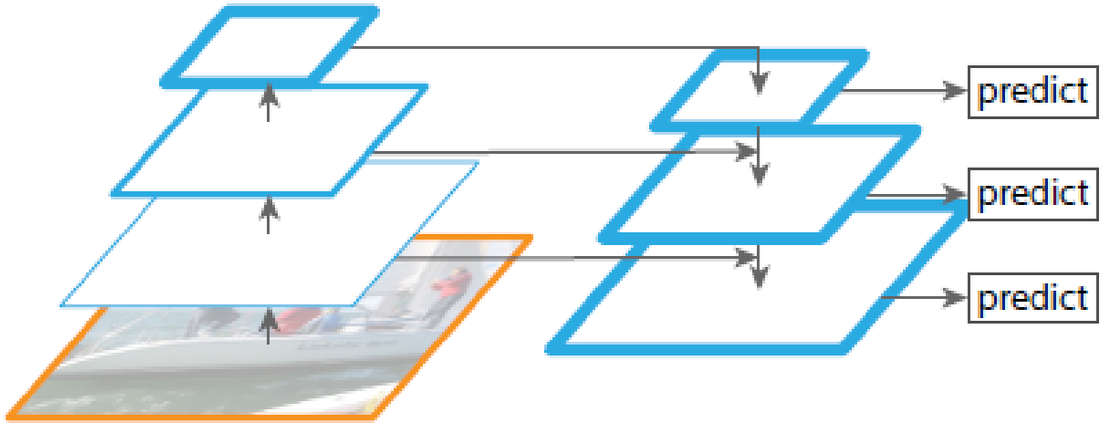
Реализовано в YOLO

Не учитывают низкоуровневую информацию

Разные архитектуры



(c) Pyramidal feature hierarchy



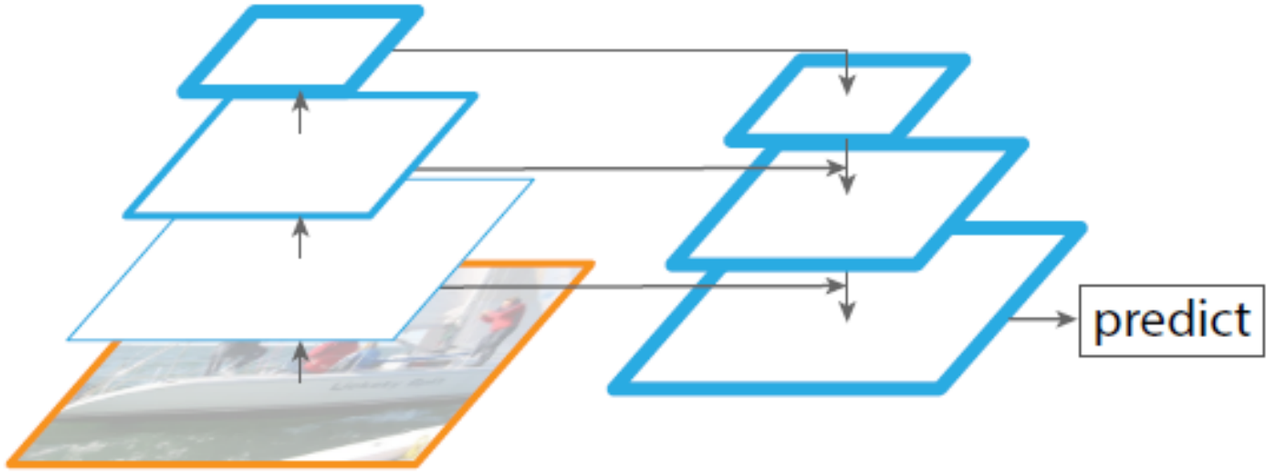
(d) Feature Pyramid Network

Реализовано в SSD

Не учитывают контекст на низком уровне

точные и быстрые

Разные архитектуры

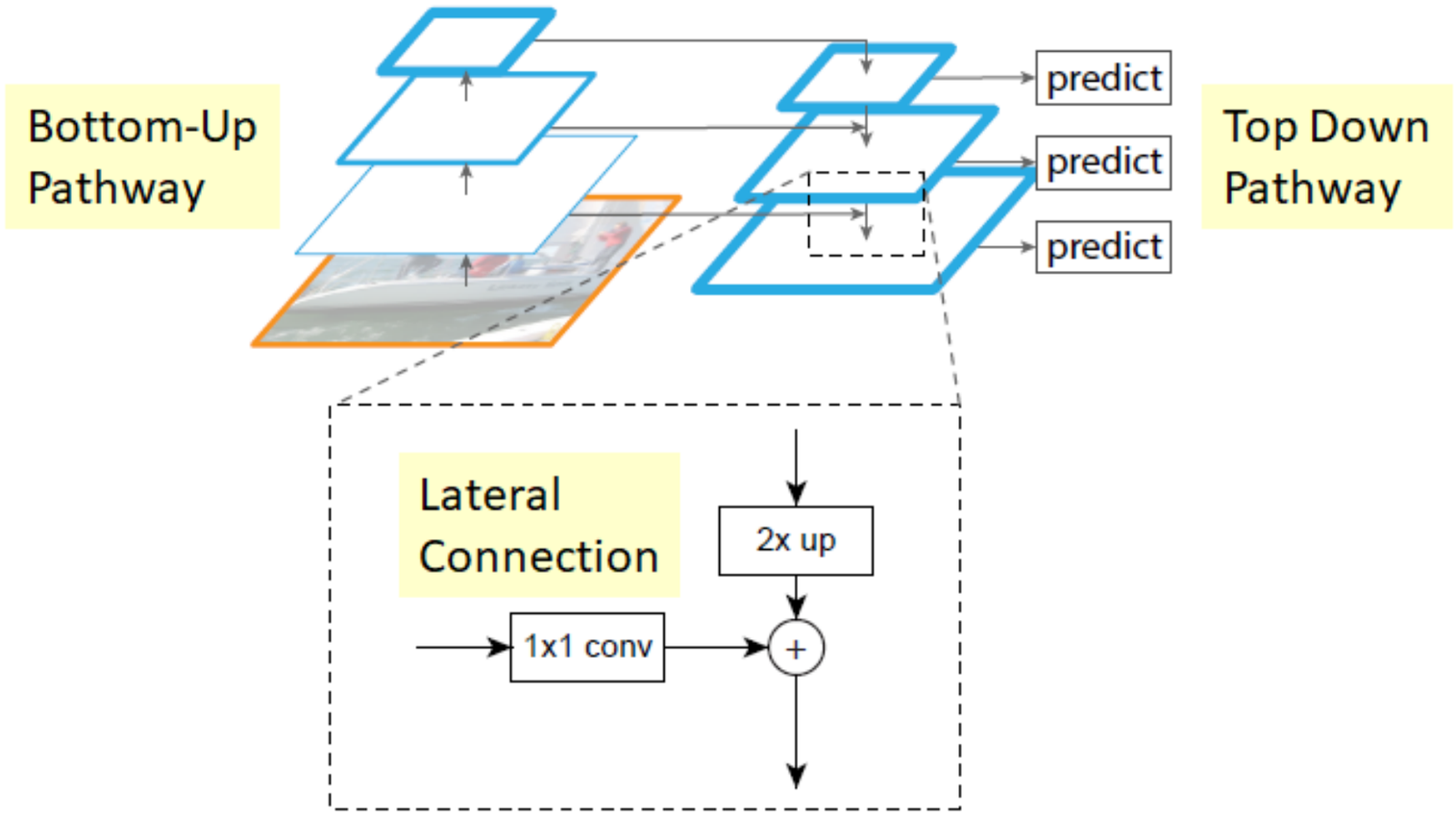


(e) Similar Structure with (d)

Иногда применяется схожая архитектура

В сегментации это U-Net

Feature Pyramid Networks (FPN)

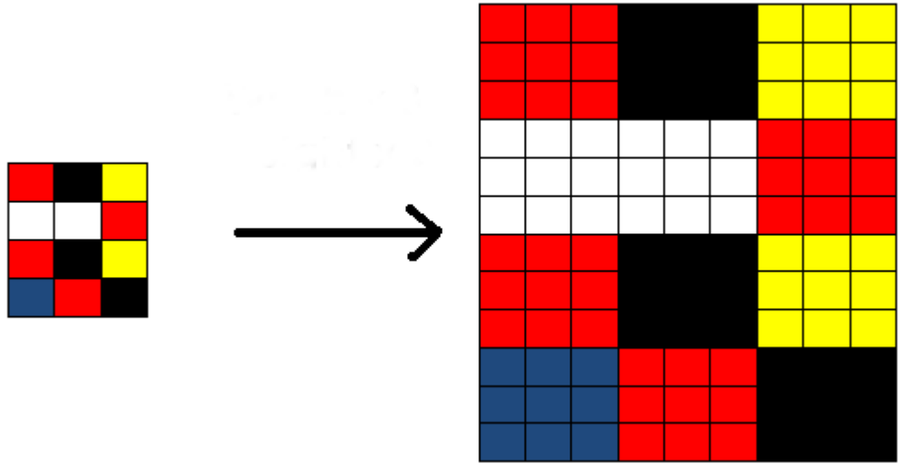


предсказания на разных уровнях

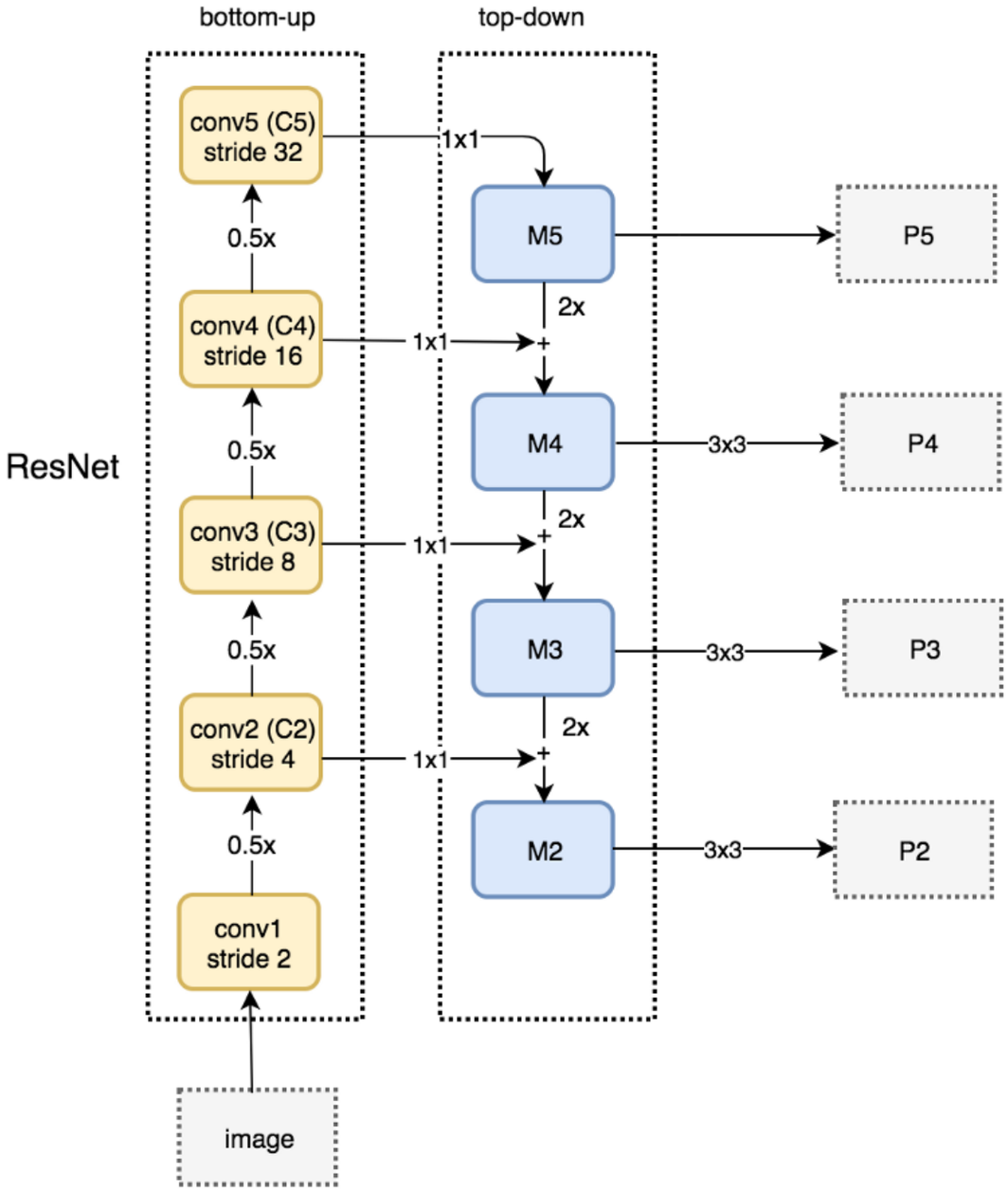
для одинаковости размера (256) свёртки 1×1

Top-down

upsampling × 2 с помощью ближайшего соседа



lateral connection складывает признаковые карты одинакового пространственного размера



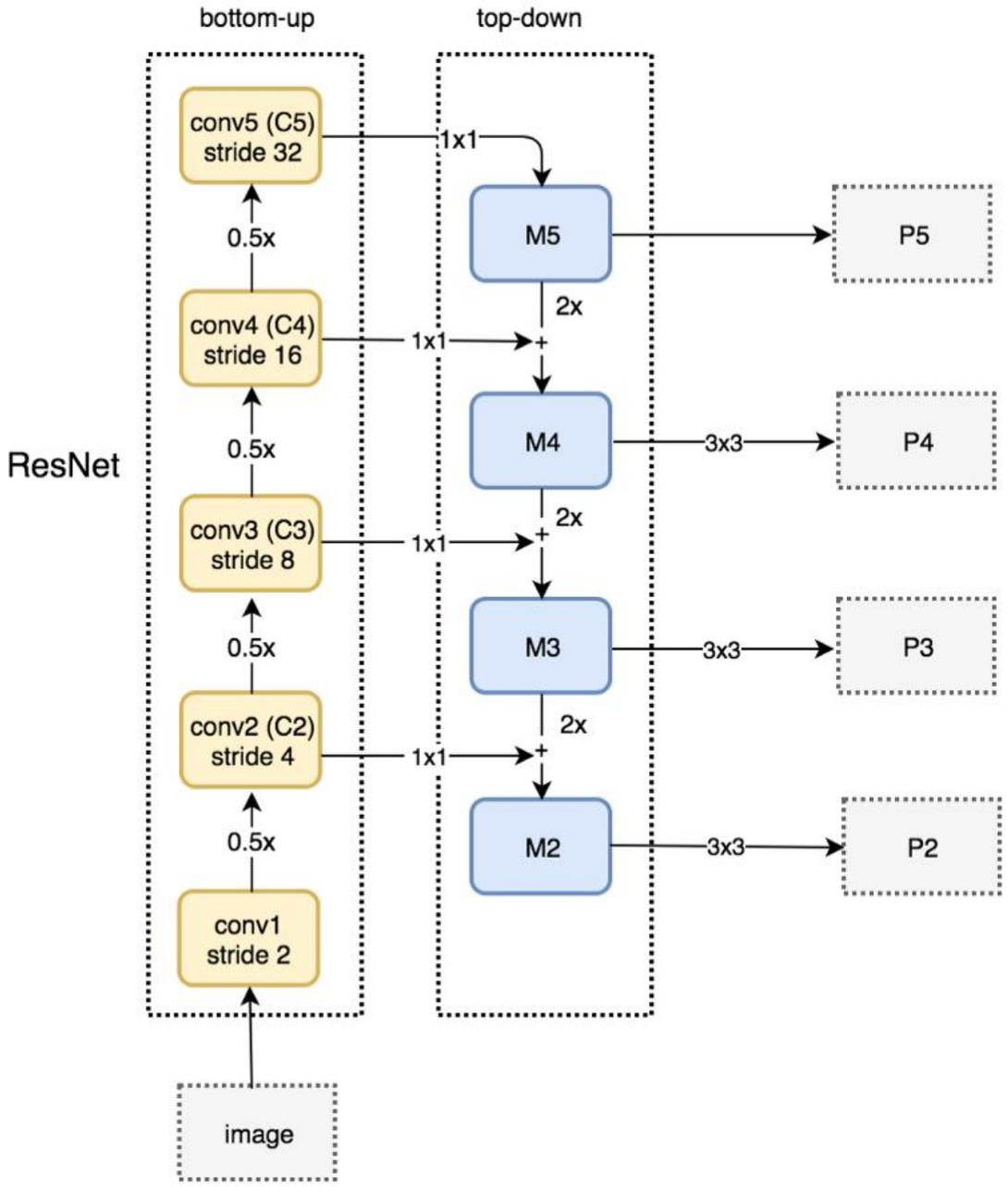
Bottom-up

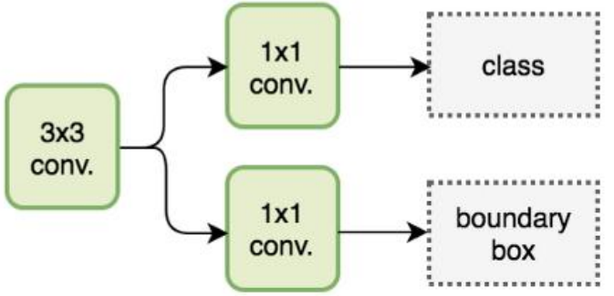
Используется ResNet

На уровнях пространственное разрешение уменьшается в 2 раза

P1 нет из-за слишком большой пространственной размерности

Это не object-detector, а построение признаков





predictor head

FPN: минутка кода

```
# из класса FPN

 def forward(self, x):
     # Bottom-up
     c1 = F.relu(self.bn1(self.conv1(x)))
     c1 = F.max_pool2d(c1, kernel_size=3,
                            stride=2, padding=1)
     c2 = self.layer1(c1)
     c3 = self.layer2(c2)
     c4 = self.layer3(c3)
     c5 = self.layer4(c4)
     # Top-down
     p5 = self.toplayer(c5)
     p4 = self._upsample_add(p5, self.latlayer1(c4))
     p3 = self._upsample_add(p4, self.latlayer2(c3))
     p2 = self._upsample_add(p3, self.latlayer3(c2))
     # Smooth
     p4 = self.smooth1(p4)
     p3 = self.smooth2(p3)
     p2 = self.smooth3(p2)
     return p2, p3, p4, p5


 def _upsample_add(self, x, y):
     _,_,H,W = y.size()
      return F.upsample(x, size=(H,W), mode='bilinear') +
                                y
```

```
def _make_layer(self, block, planes, num_blocks, stride):
    strides = [stride] + [1]*(num_blocks-1)
    layers = []
    for stride in strides:
        layers.append(block(self.in_planes, planes, stride))
        self.in_planes = planes * block.expansion
    return nn.Sequential(*layers)

def __init__(self, block, num_blocks):
    super(FPN, self).__init__()
    self.in_planes = 64

    self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=False)
    self.bn1 = nn.BatchNorm2d(64)

    # Bottom-up layers
    self.layer1 = self._make_layer(block,  64, num_blocks[0], stride=1)
    self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
    self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
    self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)

    # Top layer
    self.toplayer = nn.Conv2d(2048, 256, kernel_size=1, stride=1, padding=0)

    # Smooth layers
    self.smooth1 = nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1)
    self.smooth2 = nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1)
    self.smooth3 = nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1)

    # Lateral layers
    self.latlayer1 = nn.Conv2d(1024, 256, kernel_size=1, stride=1, padding=0)
    self.latlayer2 = nn.Conv2d( 512, 256, kernel_size=1, stride=1, padding=0)
    self.latlayer3 = nn.Conv2d( 256, 256, kernel_size=1, stride=1, padding=0)
```

https://github.com/kuangliu/pytorch-fpn/blob/master/fpn.py

FPN: минутка кода

FEATUREPYRAMIDNETWORK

CLASS torchvision.ops.FeaturePyramidNetwork(*in_channels_list: List[int], out_channels: int, extra_blocks: Optional[ExtraFPNBlock] = None, norm_layer: Optional[Callable[[...], Module]] = None*) [SOURCE]

Module that adds a FPN from on top of a set of feature maps. This is based on "Feature Pyramid Network for Object Detection".

The feature maps are currently supposed to be in increasing depth order.

The input to the model is expected to be an OrderedDict[Tensor], containing the feature maps on top of which the FPN will be added.

Parameters:

- **in_channels_list** (*list[int]*) – number of channels for each feature map that is passed to the module
- **out_channels** (*int*) – number of channels of the FPN representation
- **extra_blocks** (*ExtraFPNBlock or None*) – if provided, extra operations will be performed. It is expected to take the fpn features, the original features and the names of the original features as input, and returns a new list of feature maps and their corresponding names
- **norm_layer** (*callable, optional*) – Module specifying the normalization layer to use. Default: None

```
m = torchvision.ops.FeaturePyramidNetwork([10, 20, 30], 5)
# get some dummy data
x = OrderedDict()
x['feat0'] = torch.rand(1, 10, 64, 64)
x['feat2'] = torch.rand(1, 20, 16, 16)
x['feat3'] = torch.rand(1, 30, 8, 8)
# compute the FPN on top of x
output = m(x)
print([(k, v.shape) for k, v in output.items()])
# returns
  [('feat0', torch.Size([1, 5, 64, 64])),
   ('feat2', torch.Size([1, 5, 16, 16])),
   ('feat3', torch.Size([1, 5, 8, 8]))]
```

здесь это просто подготовка карт признаков

https://pytorch.org/vision/main/generated/torchvision.ops.FeaturePyramidNetwork.html

Feature Pyramid Networks (FPN)

До этого момента – общий способ генерации карт признаков

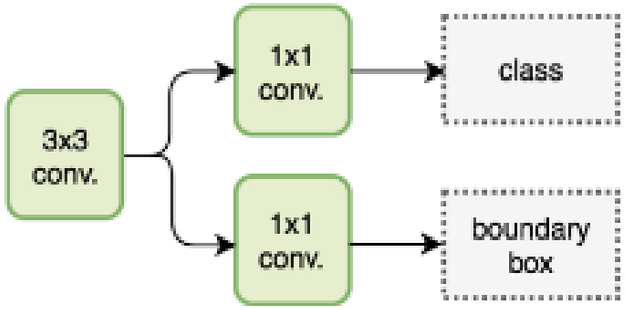
подходит для любой сети

можно использовать любые архитектуры

Применено для

- **bounding box proposal generation в RPN**
- **object detection в Fast R-CNN**

FPN + RPN (bounding box proposal generation)



3×3-свёртки на каждой карте из lateral connection для получения финальной признаковой карты

1×1-свёртки для классификаторов и регрессоров

– это называется «RPN head»

На всех уровнях одинаковые (shared) классификаторы (есть или нет объект) / регрессоры!

Они действую по отношению к набору anchors

На каждом уровне 1:2, 1:1, 2:1

по площади $\{32^2, 64^2, 128^2, 256^2, 512^2\}$ пикселей на каждом уровне

anchor позитивный, если IoU > 0.7

негативный, если IoU < 0.3

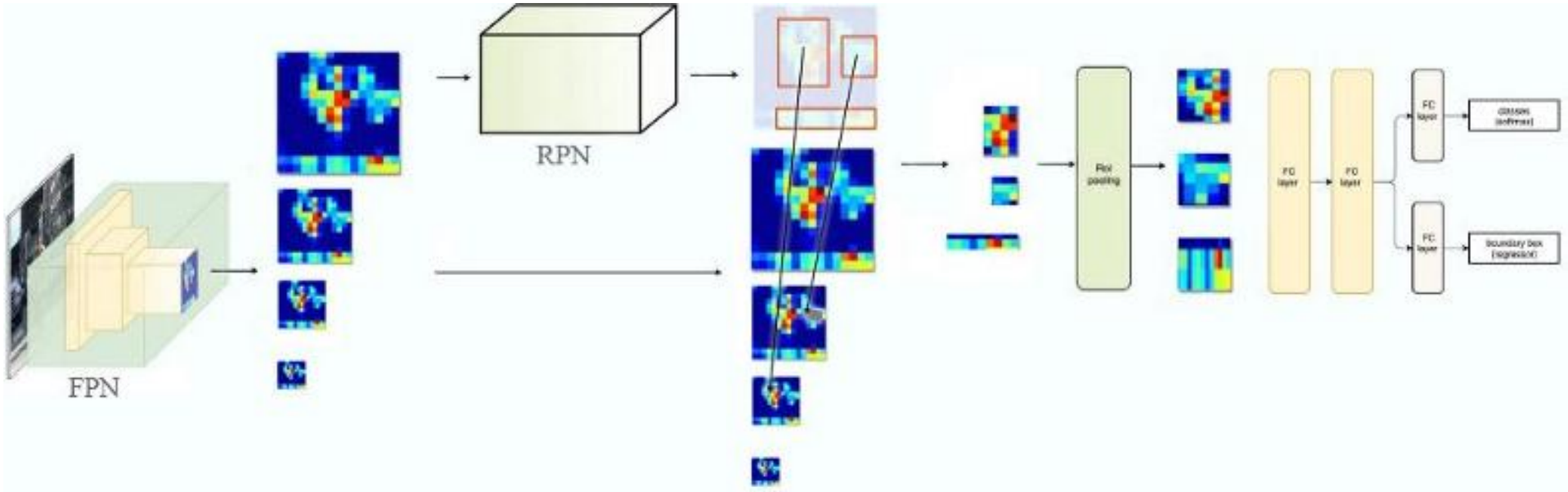
FPN + Fast R-CNN (object detection)

используем Region-of-Interest (RoI) pooling из Fast R-CNN

но есть тонкий момент – если есть регион, какому уровню его отнести?

и взять с этого уровня описание

$$k = \left\lfloor k_0 + \log_2(\sqrt{wh} / 224) \right\rfloor$$



RetinaNet

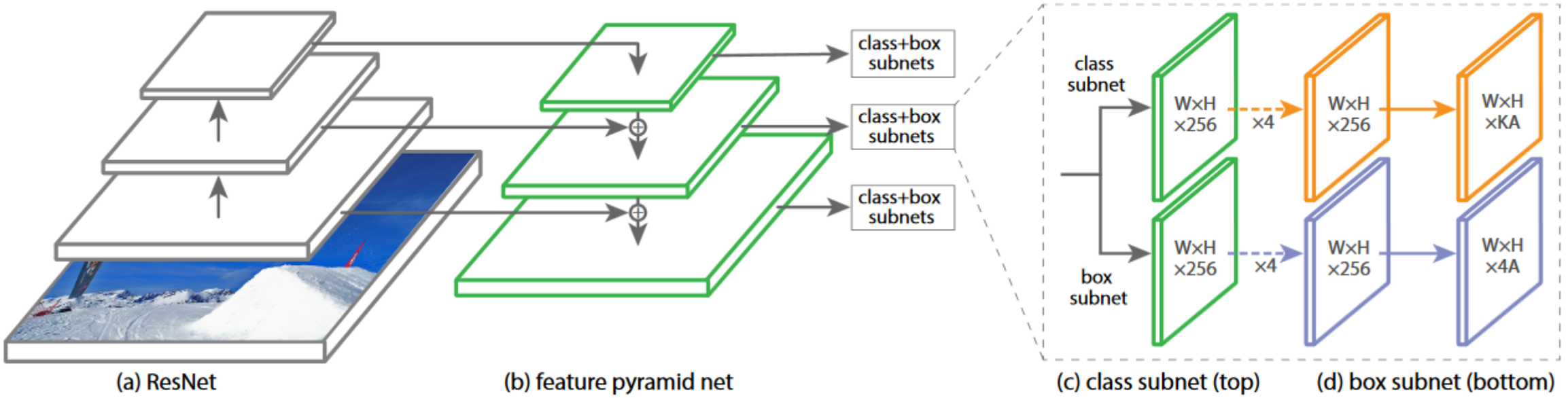
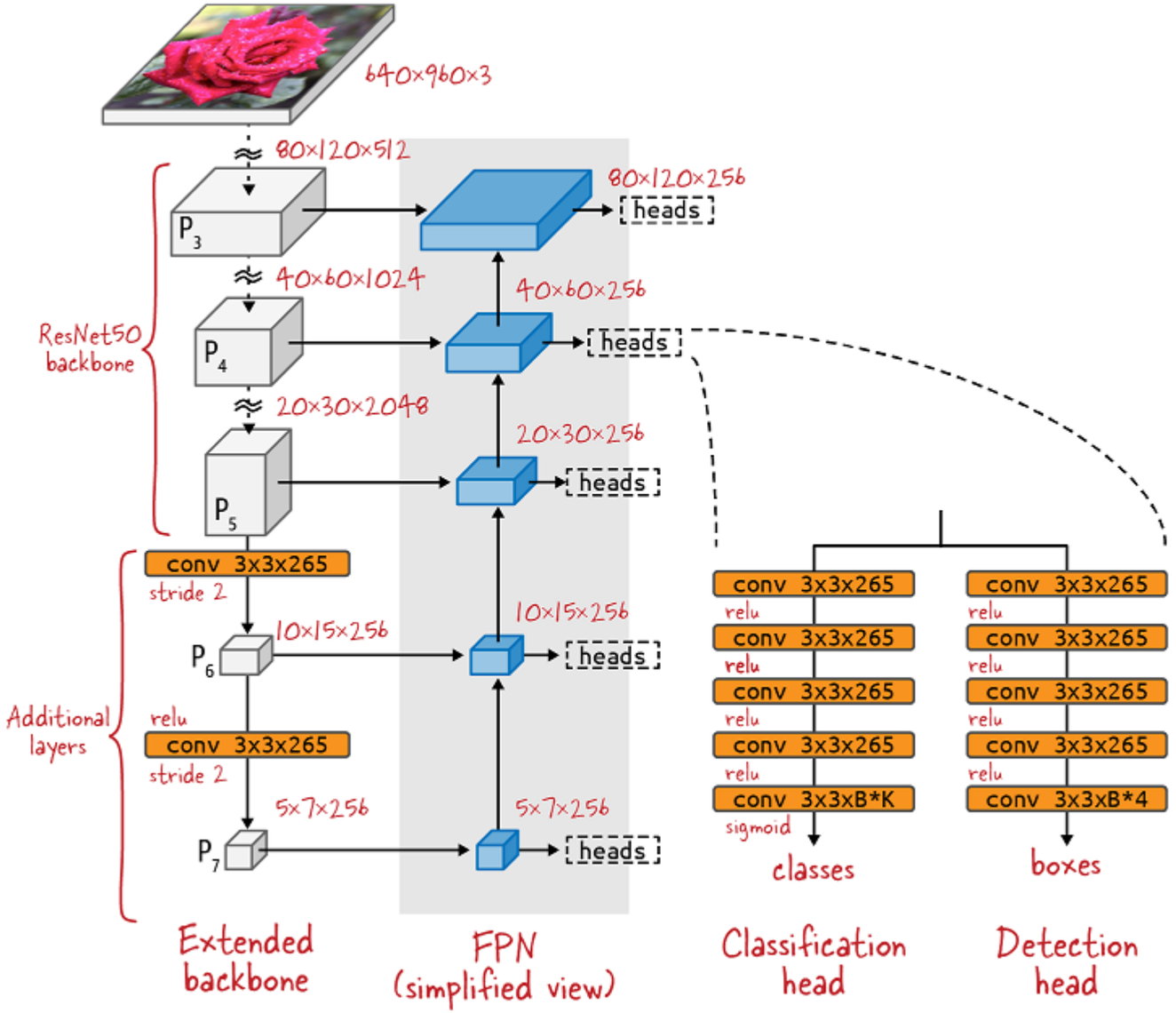


Figure 3. The one-stage **RetinaNet** network architecture uses a Feature Pyramid Network (FPN) [20] backbone on top of a feedforward ResNet architecture [16] (a) to generate a rich, multi-scale convolutional feature pyramid (b). To this backbone RetinaNet attaches two subnetworks, one for classifying anchor boxes (c) and one for regressing from anchor boxes to ground-truth object boxes (d). The network design is intentionally simple, which enables this work to focus on a novel focal loss function that eliminates the accuracy gap between our one-stage detector and state-of-the-art two-stage detectors like Faster R-CNN with FPN [20] while running at faster speeds.

one-stage detector + focal-loss, якоря (хитрые целевые значения регрессии)

Tsung-Yi Lin, et al «Focal Loss for Dense Object Detection» // https://arxiv.org/abs/1708.02002

RetinaNet



[Practical Machine Learning for Computer Vision]

RetinaNet

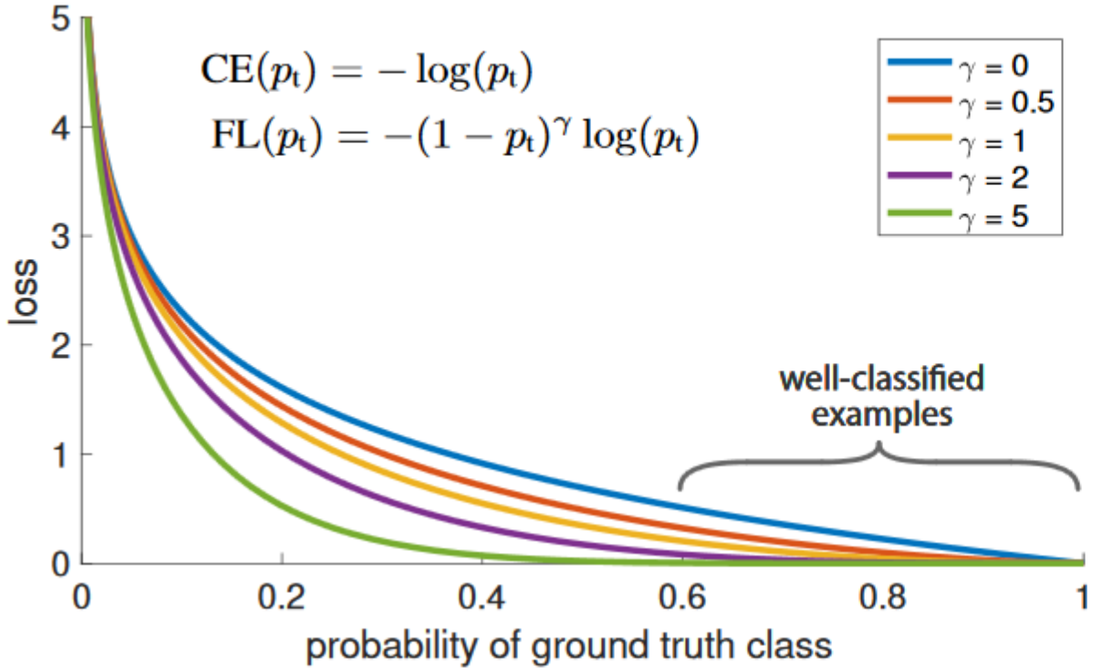
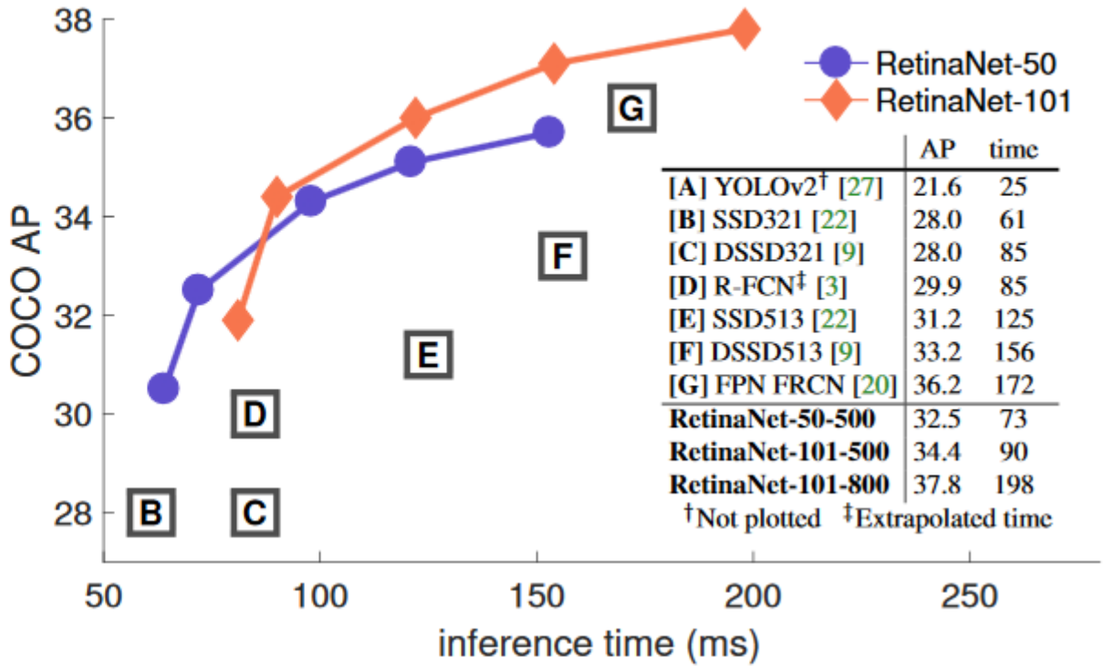


Figure 1. We propose a novel loss we term the *Focal Loss* that adds a factor $(1 - p_t)^\gamma$ to the standard cross entropy criterion. Setting $\gamma > 0$ reduces the relative loss for well-classified examples ($p_t > .5$), putting more focus on hard, misclassified examples. As our experiments will demonstrate, the proposed focal loss enables training highly accurate dense object detectors in the presence of vast numbers of easy background examples.



	AP	time
[A] YOLOv2† [27]	21.6	25
[B] SSD321 [22]	28.0	61
[C] DSSD321 [9]	28.0	85
[D] R-FCN‡ [3]	29.9	85
[E] SSD513 [22]	31.2	125
[F] DSSD513 [9]	33.2	156
[G] FPN FRCN [20]	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	37.8	198

†Not plotted ‡Extrapolated time

Figure 2. Speed (ms) versus accuracy (AP) on COCO `test-dev`. Enabled by the focal loss, our simple one-stage *RetinaNet* detector outperforms all previous one-stage and two-stage detectors, including the best reported Faster R-CNN [28] system from [20]. We show variants of RetinaNet with ResNet-50-FPN (blue circles) and ResNet-101-FPN (orange diamonds) at five scales (400-800 pixels). Ignoring the low-accuracy regime (AP<25), RetinaNet forms an upper envelope of all current detectors, and an improved variant (not shown) achieves 40.8 AP. Details are given in §5.

Anchor-Free (Proposal Free) Object Detection

Следующее поколение детекторов…
«Безъякорное детектирование»

новая прорывная идея!

anchors – очень затратны
есть гиперпараметры, от которых сильно зависит качество
все регионы-кандидаты при обучении надо размечать
(опять же – долго + гиперпараметры)

Дальше обзор разных сетей
по-разному представляют регион в каждой точке
(тут много идей взято из RPN)

Anchor-Free Object Detection

CornerNet – предсказываем «углы»

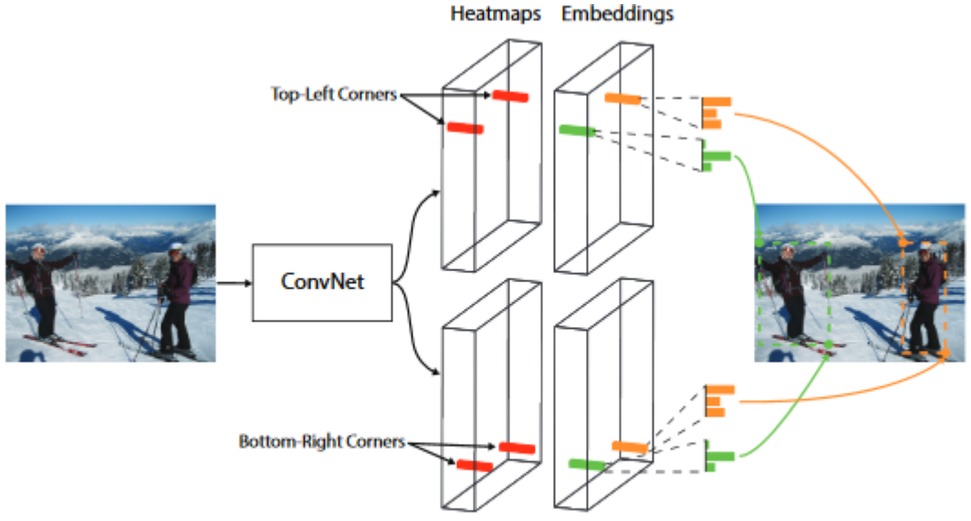


Fig. 1 We detect an object as a pair of bounding box corners grouped together. A convolutional network outputs a heatmap for all top-left corners, a heatmap for all bottom-right corners, and an embedding vector for each detected corner. The network is trained to predict similar embeddings for corners that belong to the same object.

https://arxiv.org/pdf/1808.01244.pdf

CornerNet-Lite – ускоренная версия, Начинает с уменьшенной картинки, карта внимания (attention map)

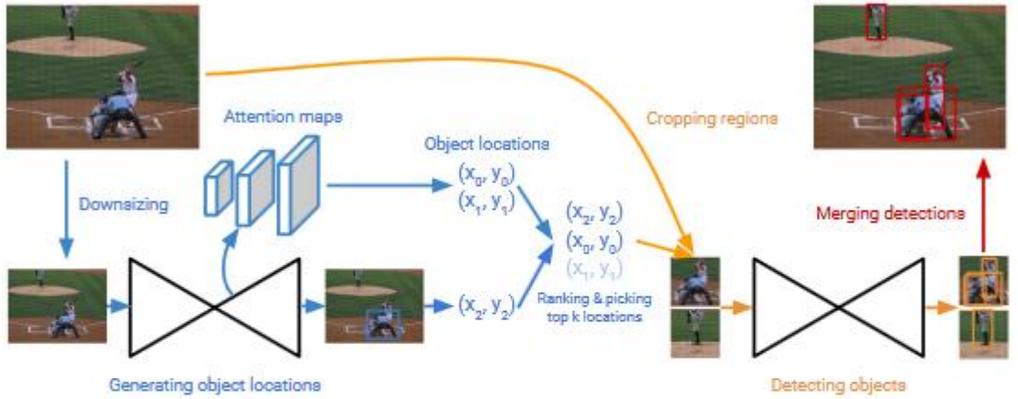
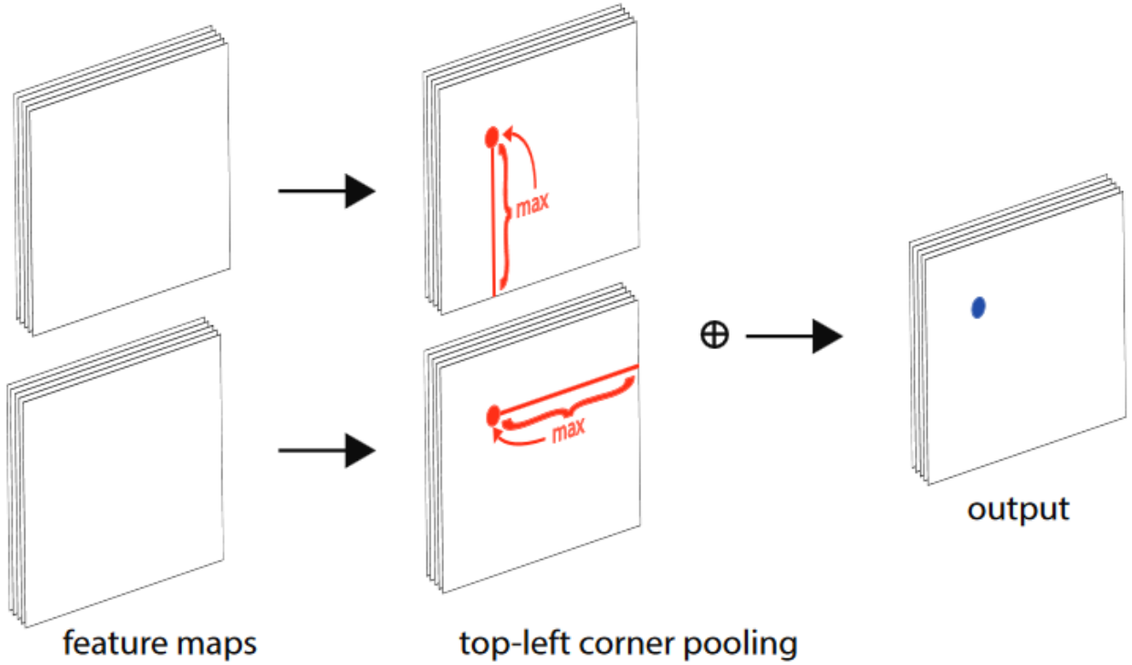


Figure 2: Overview of CornerNet-Saccade. We predict a set of possible object locations from the attention maps and bounding boxes generated on a downsized full image. We zoom into each location and crop a small region around that location. Then we detect objects in top k regions and merge the detections by NMS.

https://arxiv.org/pdf/1904.08900.pdf

немного истории этого более свежего направления

CornerNet: особый вид пулинга



т.к. в углу рамки обычно и объекта-то нет!

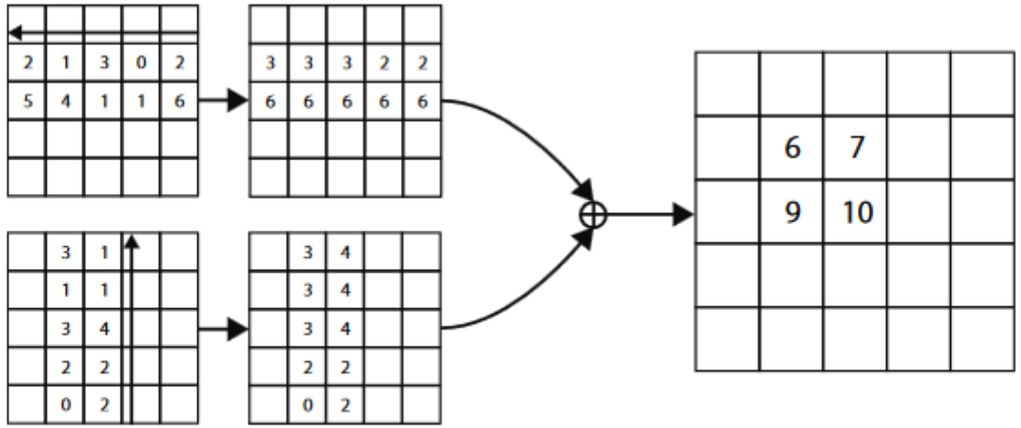


Fig. 6 The top-left corner pooling layer can be implemented very efficiently. We scan from right to left for the horizontal max-pooling and from bottom to top for the vertical max-pooling. We then add two max-pooled feature maps.

CornerNet: особый вид пулинга

Fig. 8 Qualitative examples showing corner pooling helps better localize the corners.

CornerNet: выход сети

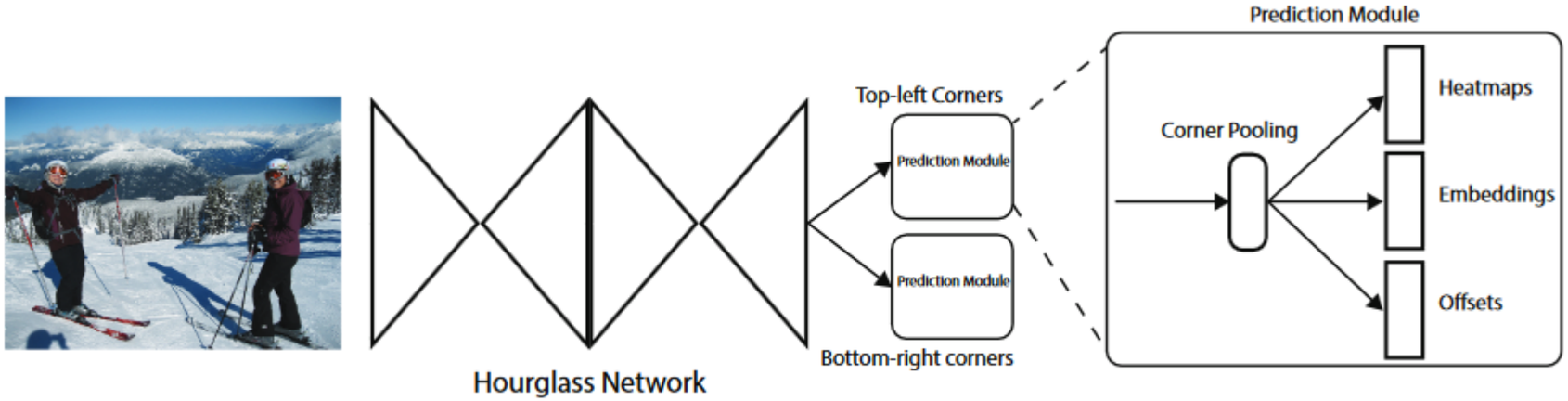


Fig. 4 Overview of CornerNet. The backbone network is followed by two prediction modules, one for the top-left corners and the other for the bottom-right corners. Using the predictions from both modules, we locate and group the corners.

для «top-left» и «bottom-right» предсказываем:

heatmaps (вероятности быть углом для каждого из классов) тут нет класса «фон»

embedding: расстояния между представлениями углов одной рамки минимально

offsets – для редактирования координат углов

в каждом модуле (tl и br) свой corner pooling module

тут нет признаков с разных масштабов

CornerNet: обучение

Fig. 5 "Ground-truth" heatmaps for training. Boxes *(green dotted rectangles)* whose corners are within the radii of the positive locations *(orange circles)* still have large overlaps with the ground-truth annotations *(red solid rectangles)*.

Позитивные примеры – истинные углы, но также считаем позитивным всё, что недалеко от углов

focal loss

Anchor-Free Object Detection

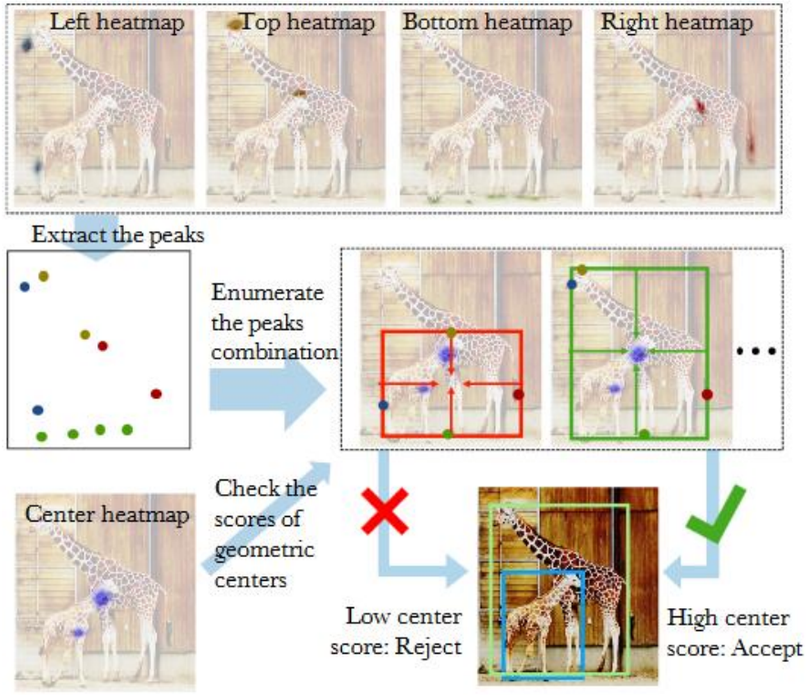


Figure 2: Illustration of our object detection method. Our network predicts four extreme point heatmaps (Top. We shown the heatmap overlaid on the input image) and one center heatmap (Bottom row left) for each category. We enumerate the combinations of the peaks (Middle left) of four extreme point heatmaps and compute the geometric center of the composed bounding box (Middle right). A bounding box is produced if and only if its geometric center has a high response in the center heatmap (Bottom right).

ExtremeNet

Предсказываются 4 экстремальные точки (самая верхняя и т.п.) и центр, из них группируются ответы, если 4 точкам определённого класса соответствует центр этого класса, то формируется рамка-ответ

Xingyi Zhou et al. «Bottom-up Object Detection by Grouping Extreme and Center Points» // https://arxiv.org/pdf/1901.08043.pdf

Anchor-Free Object Detection: CenterNet

Беъзякорный подход можно применять для разных задач – меняется, что предсказываем

НС: $\mathbb{R}^{w\times h\times 3} \to \mathbb{R}^{\frac{w}{k}\times\frac{h}{k}\times c}$

C **– сколько параметров надо определять**

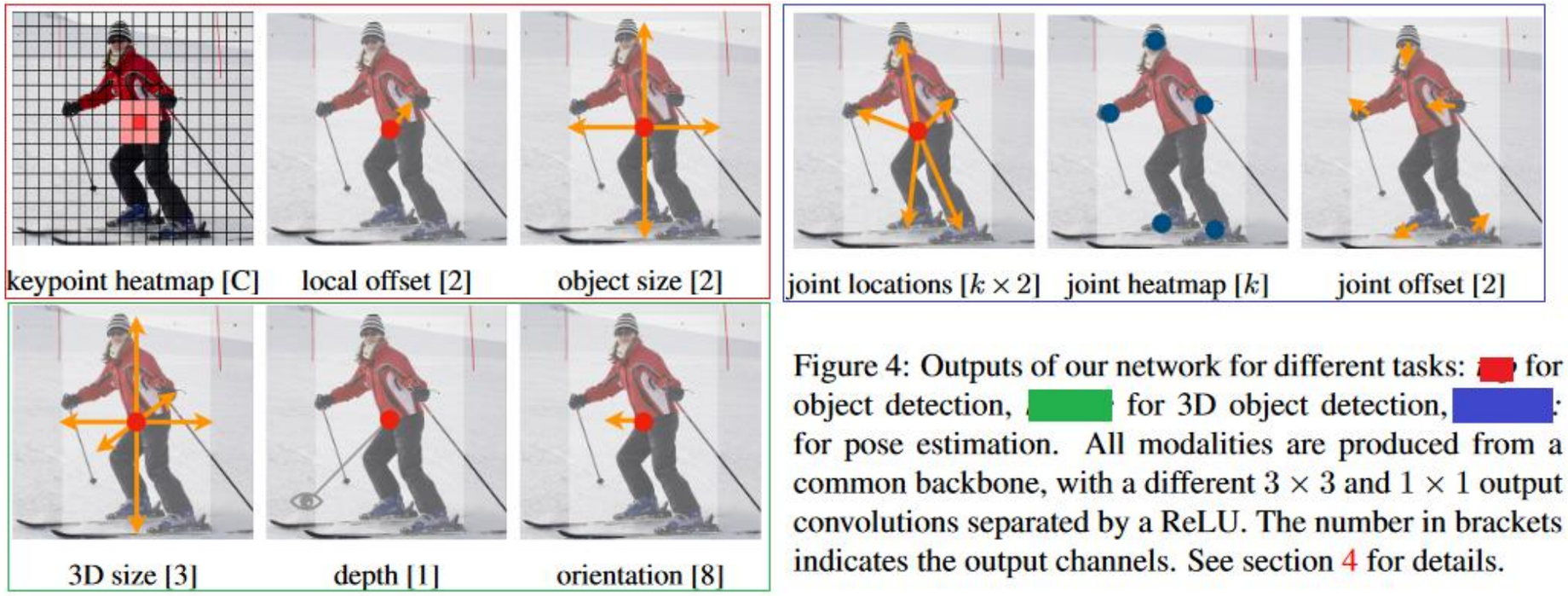
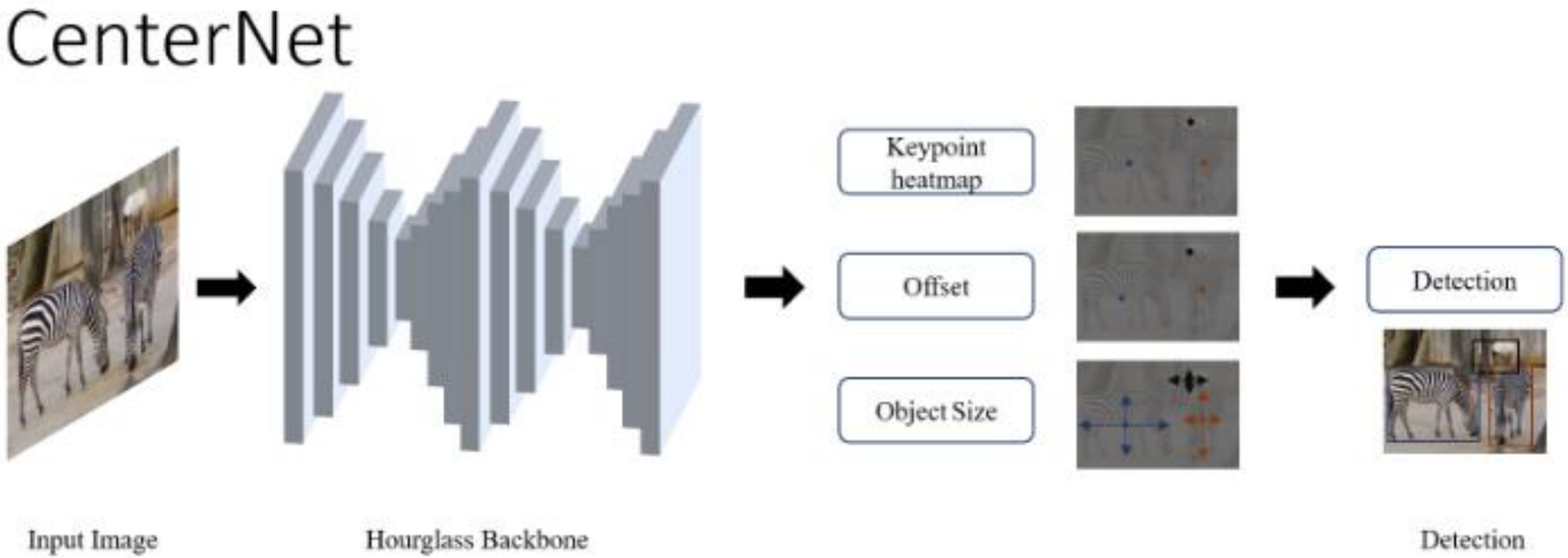


Figure 4: Outputs of our network for different tasks: for object detection, for 3D object detection, for pose estimation. All modalities are produced from a common backbone, with a different 3×3 and 1×1 output convolutions separated by a ReLU. The number in brackets indicates the output channels. See section 4 for details.

Xingyi Zhou et al. «Objects as Points» // https://arxiv.org/pdf/1904.07850.pdf

Anchor-Free Object Detection: CenterNet



на базе Hourglass-101

Anchor-Free Object Detection

CenterNet-D **Идея похожа на ExtremeNet**

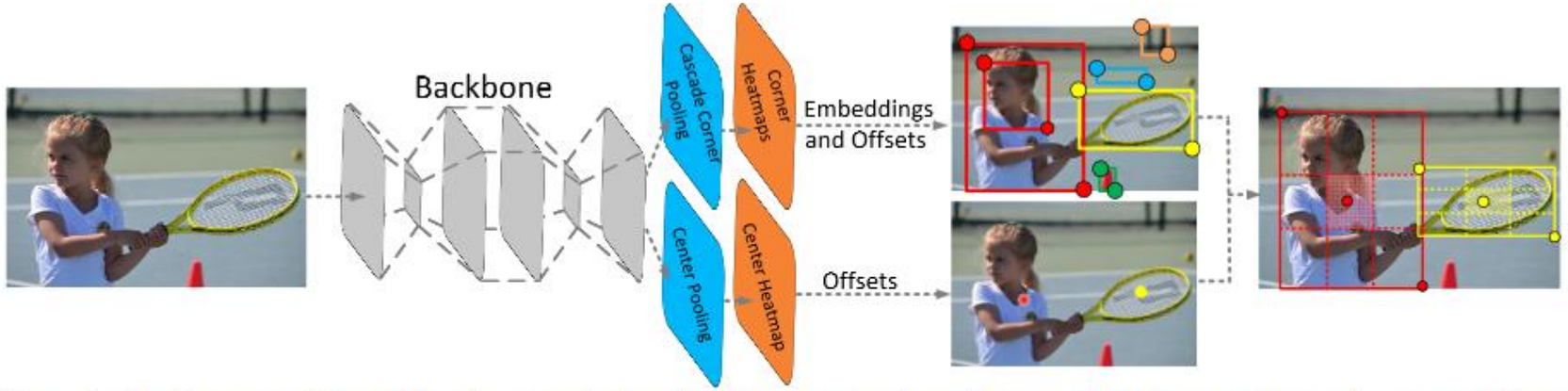


Figure 2: Architecture of CenterNet. A convolutional backbone network applies cascade corner pooling and center pooling to output two corner heatmaps and a center keypoint heatmap, respectively. Similar to CornerNet, a pair of detected corners and the similar embeddings are used to detect a potential bounding box. Then the detected center keypoints are used to determine the final bounding boxes.

https://arxiv.org/pdf/1904.08189.pdf

Anchor-Free Object Detection

FSAF: Feature Selective Anchor-Free Module for Single-Shot Object Detection

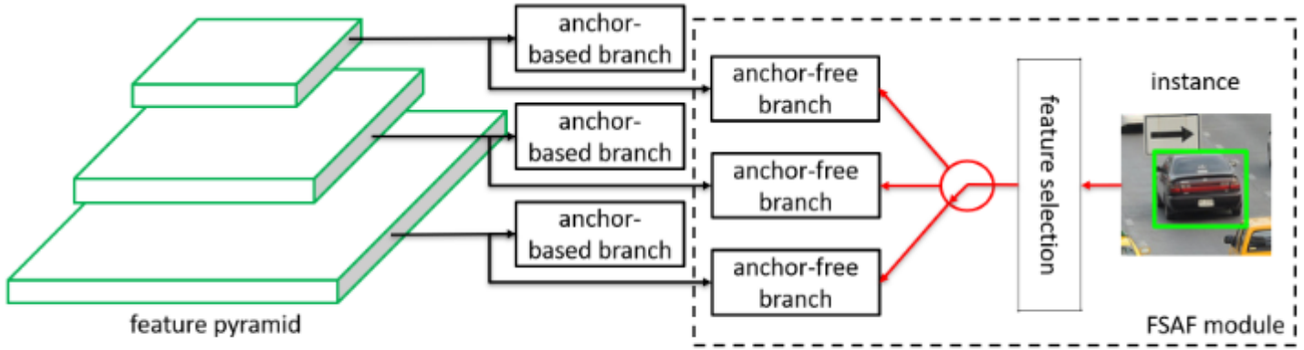


Figure 3: Overview of our FSAF module plugged into conventional anchor-based detection methods. During training, each instance is assigned to a pyramid level via feature selection for setting up supervision signals.

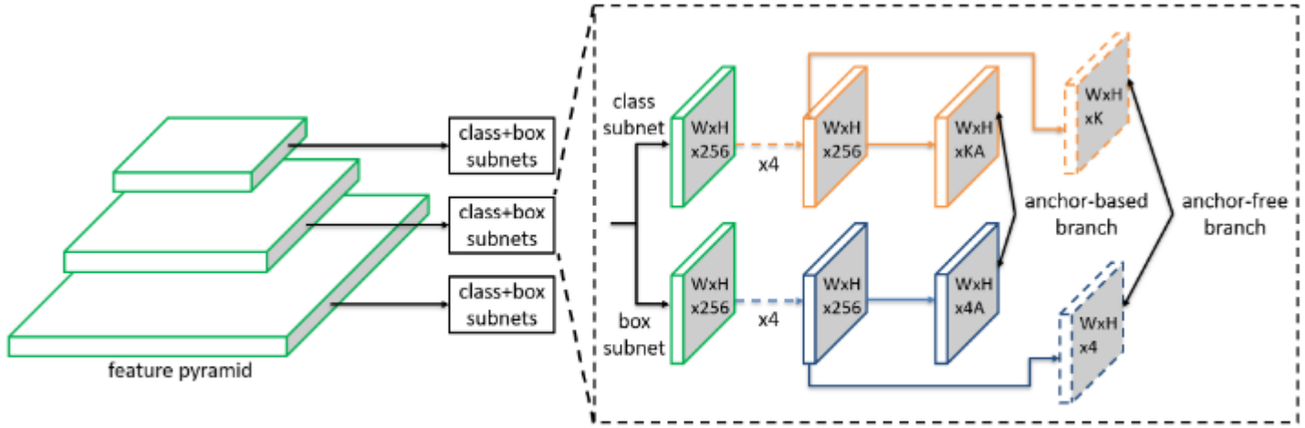


Figure 4: Network architecture of RetinaNet with our FSAF module. The FSAF module only introduces two additional conv layers (dashed feature maps) per pyramid level, keeping the architecture fully convolutional.

anchor-free + anchor-based, RetinaNet + FSAF

Anchor-Free Object Detection

Есть также

FoveaBox // https://arxiv.org/abs/1904.03797

FCOS

(ниже)

RepPoints // https://arxiv.org/abs/1904.11490

FCOS: Fully Convolutional One-Stage Object Detection

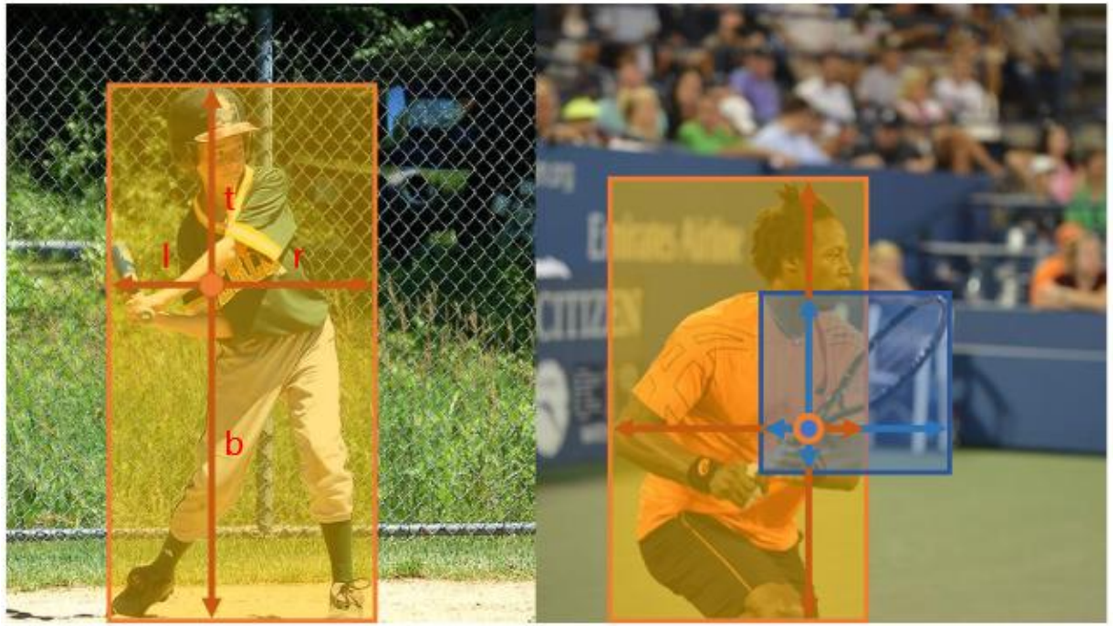


Figure 1 – As shown in the left image, FCOS works by predicting a 4D vector (l, t, r, b) encoding the location of a bounding box at each foreground pixel (supervised by ground-truth bounding box information during training). The right plot shows that when a location residing in multiple bounding boxes, it can be ambiguous in terms of which bounding box this location should regress.

есть целевые регионы – им сразу и обучаемся

«One-Stage» – нет как раньше отдельных подзадач

для любой точки на изображении:
если она попадает в целевой регион – «позитивная»
если в несколько – «двузначная»
(относим к минимальному)

основа: ResNet-50

Zhi Tian, Chunhua Shen, Hao Chen, Tong He «FCOS: Fully Convolutional One-Stage Object Detection» // https://arxiv.org/pdf/1904.01355.pdf

FCOS: Fully Convolutional One-Stage Object Detection

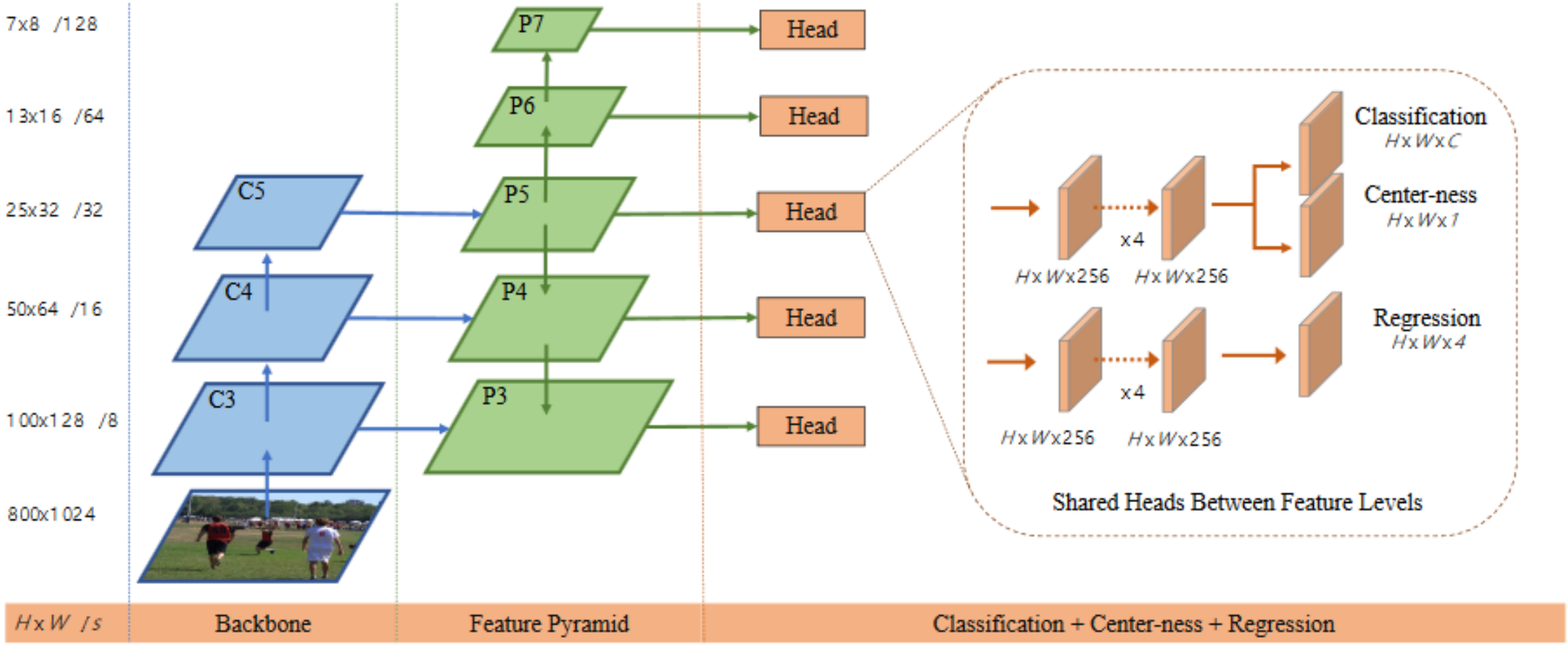


Figure 2 – The network architecture of FCOS, where C3, C4, and C5 denote the feature maps of the backbone network and P3 to P7 are the feature levels used for the final prediction. $H \times W$ is the height and width of feature maps. '/s' ($s = 8, 16, ..., 128$) is the down-sampling ratio of the feature maps at the level to the input image. As an example, all the numbers are computed with an 800×1024 input.

Feature Pyramid Network(FPN) + выходы для центра, регрессии и класса

FCOS: Center-ness

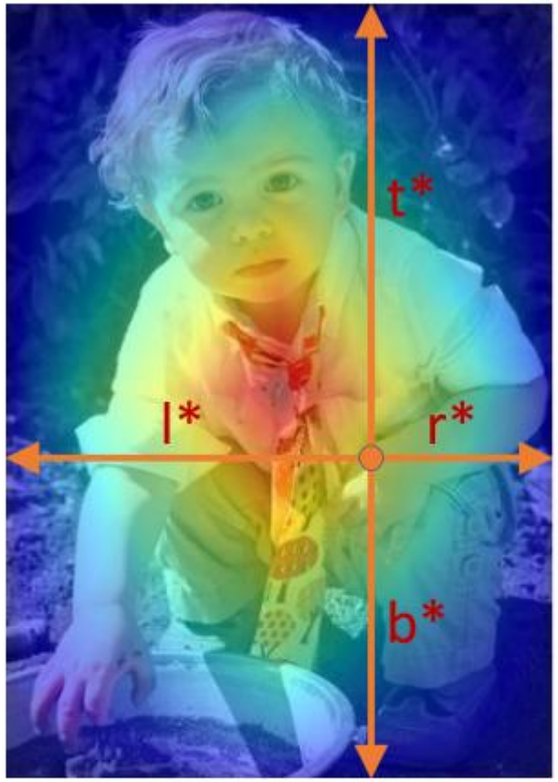


Figure 3 – Center-ness. Red, blue, and other colors denote 1, 0 and the values between them, respectively. Center-ness is computed by Eq. (3) and decays from 1 to 0 as the location deviates from the center of the object. When testing, the center-ness predicted by the network is multiplied with the classification score thus can down-weight the low-quality bounding boxes predicted by a location far from the center of an object.

В каждой точке предсказывается (l, t, r, b),

а не (x, y, w, h)

+ класс (метка из C) + центральность (число)

$$\text{centerness}^* = \sqrt{\frac{\min(l^*, r^*)}{\max(l^*, r^*)} \times \frac{\min(t^*, b^*)}{\max(t^*, b^*)}}.$$

center-ness на [0, 1] обучается с помощью binary cross entropy (BCE) loss

при работе к точкам с большим значением можно применить NMS

В классификации вместо softmax – мультикласс

FCOS: детали

В регрессии exp(*s*•x), т.к. значения положительны
(коэффициент зависит от уровня P3, ..., P7)

focal loss + IoU loss (из предыдущей работы)

не на всех уровнях (P3, ..., P7) надо детектировать регионы!
Если целевые значения регрессии превышают порог (слишком большой регион),
то считаем, что тут нет региона

центральность умножается на вероятность класса при отборе регионов
(это убрало «плохие регионы» и стало лучше предыдущих методов)
и можно только центральные точки использовать как положительные примеры
(это тоже улучшает качество)

FCOS: Fully Convolutional One-Stage Object Detection

Method	Backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods:							
Faster R-CNN w/ FPN [14]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [11]	Inception-ResNet-v2 [27]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w/ TDM [25]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	52.1
One-stage methods:							
YOLOv2 [22]	DarkNet-19 [22]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [18]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [5]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [15]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
CornerNet [13]	Hourglass-104	40.5	56.5	43.1	19.4	42.7	53.9
FSAF [34]	ResNeXt-64x4d-101-FPN	42.9	63.8	46.3	26.6	46.2	52.7
FCOS	ResNet-101-FPN	41.5	60.7	45.0	24.4	44.8	51.6
FCOS	HRNet-W32-5l [26]	42.0	60.4	45.3	25.4	45.0	51.0
FCOS	ResNeXt-32x8d-101-FPN	42.7	62.2	46.1	26.0	45.6	52.6
FCOS	ResNeXt-64x4d-101-FPN	43.2	62.8	46.6	26.5	46.2	53.3
FCOS w/ improvements	ResNeXt-64x4d-101-FPN	**44.7**	**64.1**	**48.4**	**27.6**	**47.5**	**55.6**

Table 5 – FCOS vs. other state-of-the-art two-stage or one-stage detectors (*single-model and single-scale results*). FCOS outperforms the anchor-based counterpart RetinaNet by 2.4% in AP with the same backbone. FCOS also outperforms the recent anchor-free one-stage detector CornerNet with much less design complexity. Refer to Table 3 for details of "improvements".

DEtection TRansformer (DETR)

пока не было трансформера – помогает анализировать попарные взаимодействия

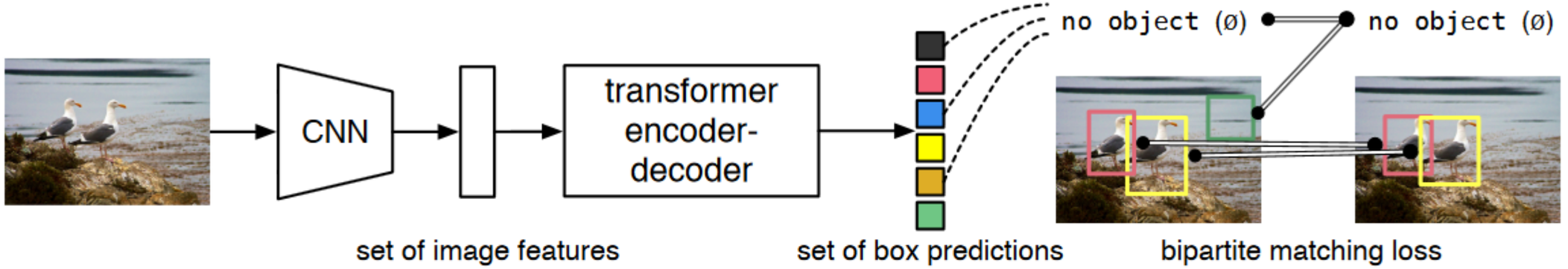


Fig. 1: DETR directly predicts (in parallel) the final set of detections by combining a common CNN with a transformer architecture. During training, bipartite matching uniquely assigns predictions with ground truth boxes. Prediction with no match should yield a "no object" (∅) class prediction.

нет каких-то специальных слоёв (могут использоваться стандартные библиотеки)

transformer encoder-decoder

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, Sergey Zagoruyko «End-to-End Object Detection with Transformers» // https://arxiv.org/pdf/2005.12872.pdf

DEtection TRansformer (DETR)

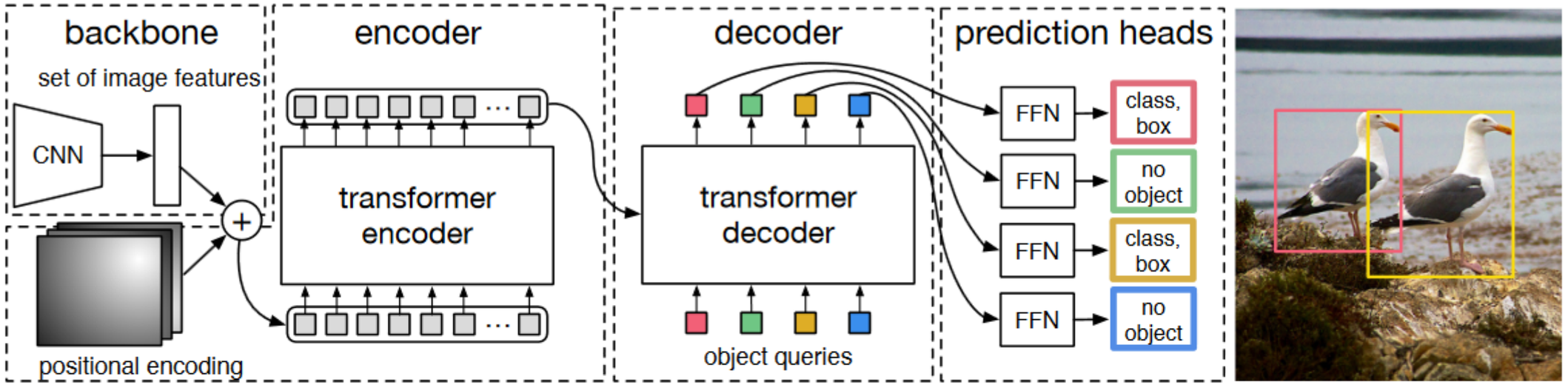


Fig. 2: DETR uses a conventional CNN backbone to learn a 2D representation of an input image. The model flattens it and supplements it with a positional encoding before passing it into a transformer encoder. A transformer decoder then takes as input a small fixed number of learned positional embeddings, which we call *object queries*, and additionally attends to the encoder output. We pass each output embedding of the decoder to a shared feed forward network (FFN) that predicts either a detection (class and bounding box) or a "`no object`" class.

делается N=100 предсказаний (это число определяет размер входа в декодере)

DEtection TRansformer (DETR)

по венгерскому алгоритму – оптимальное попарное соответствие прогноз-истина

N ↔ M>N

$$\hat{\sigma} = \arg\min_{\sigma \in \mathfrak{S}_N} \sum_i^N \mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)})$$

$$\mathcal{L}_{\text{Hungarian}}(y, \hat{y}) = \sum_{i=1}^{N} \left[-\log \hat{p}_{\hat{\sigma}(i)}(c_i) + \mathbb{1}_{\{c_i \neq \varnothing\}} \mathcal{L}_{\text{box}}(b_i, \hat{b}_{\hat{\sigma}}(i)) \right]$$

backbone (ImageNet pretrained ResNet-50): $\mathbb{R}^{3\times H\times W} \rightarrow \mathbb{R}^{2048\times(H/32)\times(W/32)}$

encoder – кодирует клетки

decoder – параллельно делает предсказания для кандидатов ~~(есть их positional encodings)~~ подсматривая в коды клеток

object queries – это обучаемые представления-входы (их фиксированное число N)

FFN получает классы и координаты (NMS не нужен)

DEtection TRansformer (DETR)

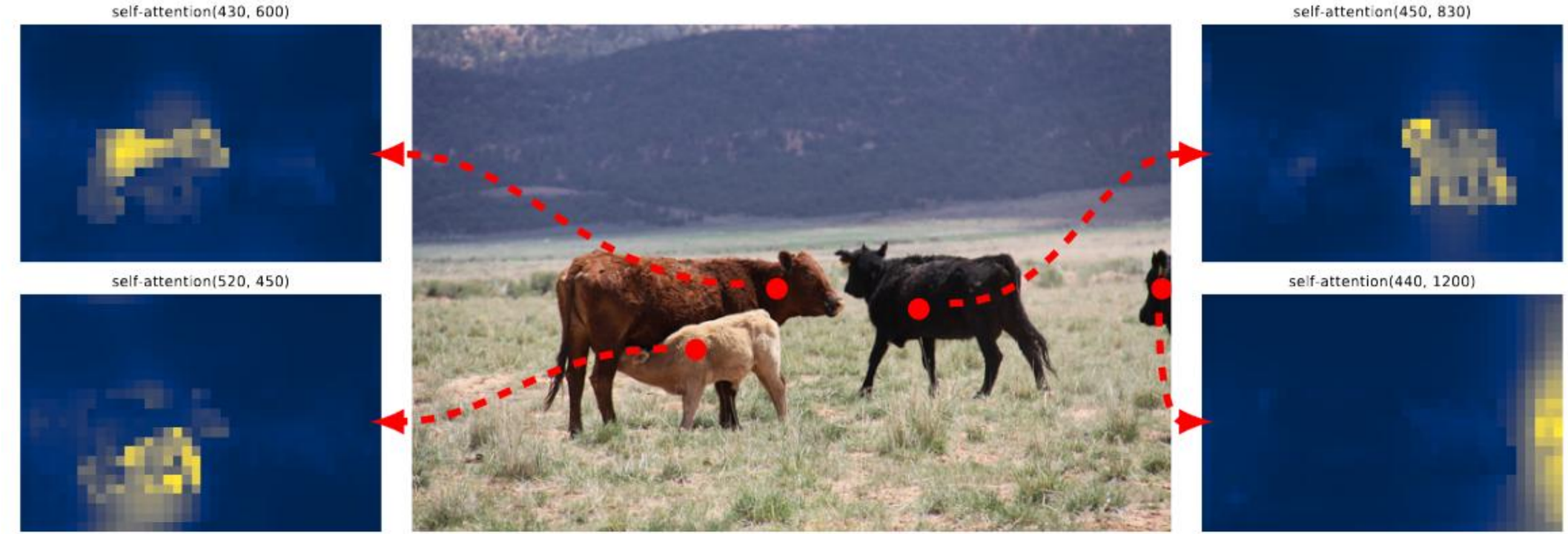


Fig. 3: Encoder self-attention for a set of reference points. The encoder is able to separate individual instances. Predictions are made with baseline DETR model on a validation set image.

DEtection TRansformer (DETR)

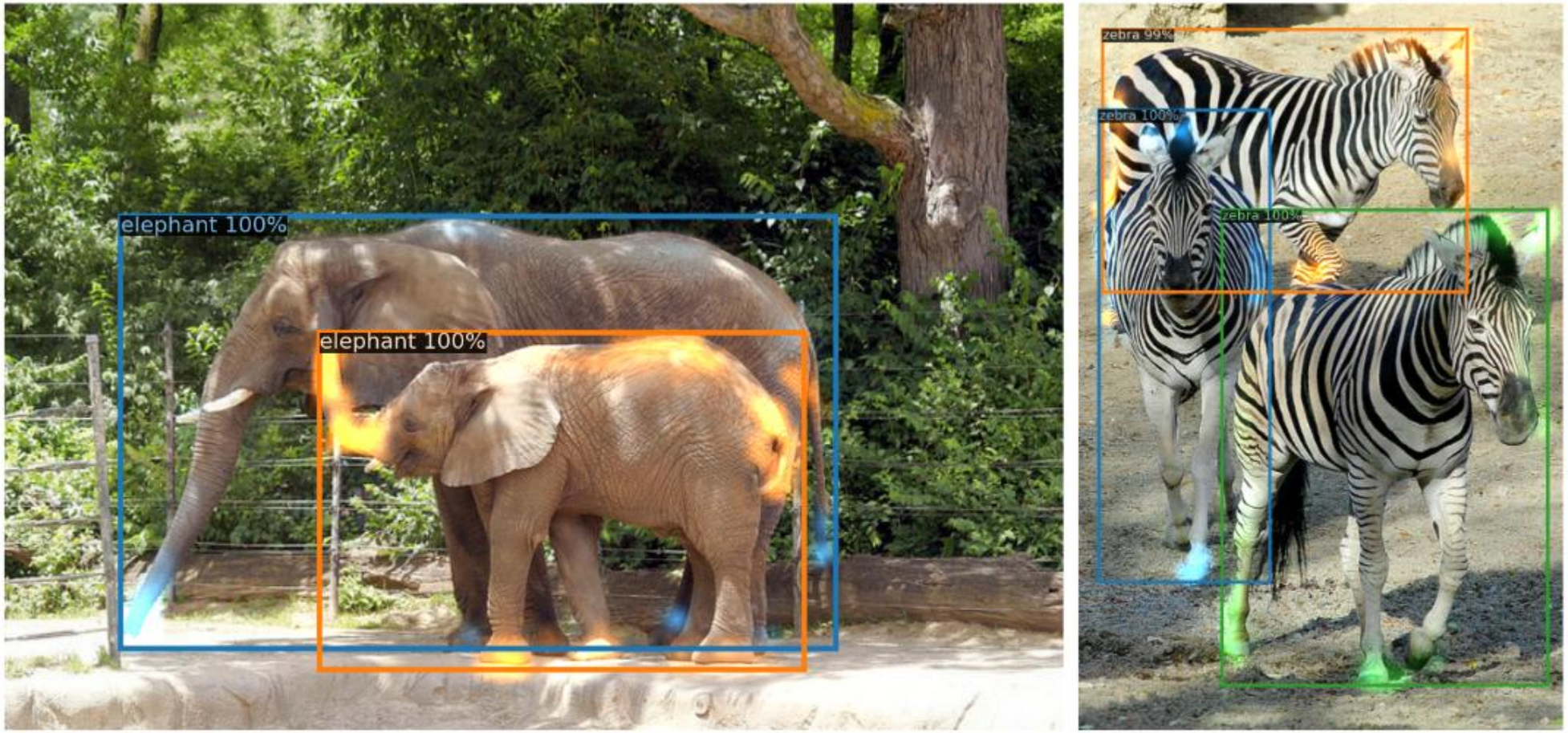


Fig. 6: Visualizing decoder attention for every predicted object (images from COCO `val` set). Predictions are made with DETR-DC5 model. Attention scores are coded with different colors for different objects. Decoder typically attends to object extremities, such as legs and heads. Best viewed in color.

DEtection TRansformer (DETR)

Интересные находки

0) неавторегрессионная модель! Число входных токенов N=100 фиксировано!

1) NMS можно не использовать:
учим только в одном токене предсказывать каждый объект.
Из-за механизма внимания это удаётся

2) в кодировщике можно не использовать кодирование позиций,
В декодировщике – надо!

3) не удалось упростить трансформер
(от чего-то избавиться)

4) есть разные доработки DETR
(например, с focal loss)

Минутка кода: DETR (упрощённая версия)

```
import torch
from torch import nn
from torchvision.models import resnet50

class DETR(nn.Module):
    def __init__(self, num_classes, hidden_dim, nheads, num_encoder_layers, num_decoder_layers):
        super().__init__()
        self.backbone = nn.Sequential(*list(resnet50(pretrained=True).children())[:-2])
        self.conv = nn.Conv2d(2048, hidden_dim, 1)
        self.transformer = nn.Transformer(hidden_dim, nheads,
        num_encoder_layers, num_decoder_layers)
        self.linear_class = nn.Linear(hidden_dim, num_classes + 1)
        self.linear_bbox = nn.Linear(hidden_dim, 4)
        self.query_pos = nn.Parameter(torch.rand(100, hidden_dim))
        self.row_embed = nn.Parameter(torch.rand(50, hidden_dim // 2))
        self.col_embed = nn.Parameter(torch.rand(50, hidden_dim // 2))
    def forward(self, inputs):
        x = self.backbone(inputs)
        h = self.conv(x)
        H, W = h.shape[-2:]
        pos = torch.cat([self.col_embed[:W].unsqueeze(0).repeat(H, 1, 1),
                         self.row_embed[:H].unsqueeze(1).repeat(1, W, 1),], dim=-1).flatten(0, 1).unsqueeze(1)
        h = self.transformer(pos + h.flatten(2).permute(2, 0, 1), self.query_pos.unsqueeze(1))
        return self.linear_class(h), self.linear_bbox(h).sigmoid()

detr = DETR(num_classes=91, hidden_dim=256, nheads=8, num_encoder_layers=6, num_decoder_layers=6)
detr.eval()
inputs = torch.randn(1, 3, 800, 1200)
logits, bboxes = detr(inputs)
```

SotA 2021-22 (COCO): приёмы аугментации

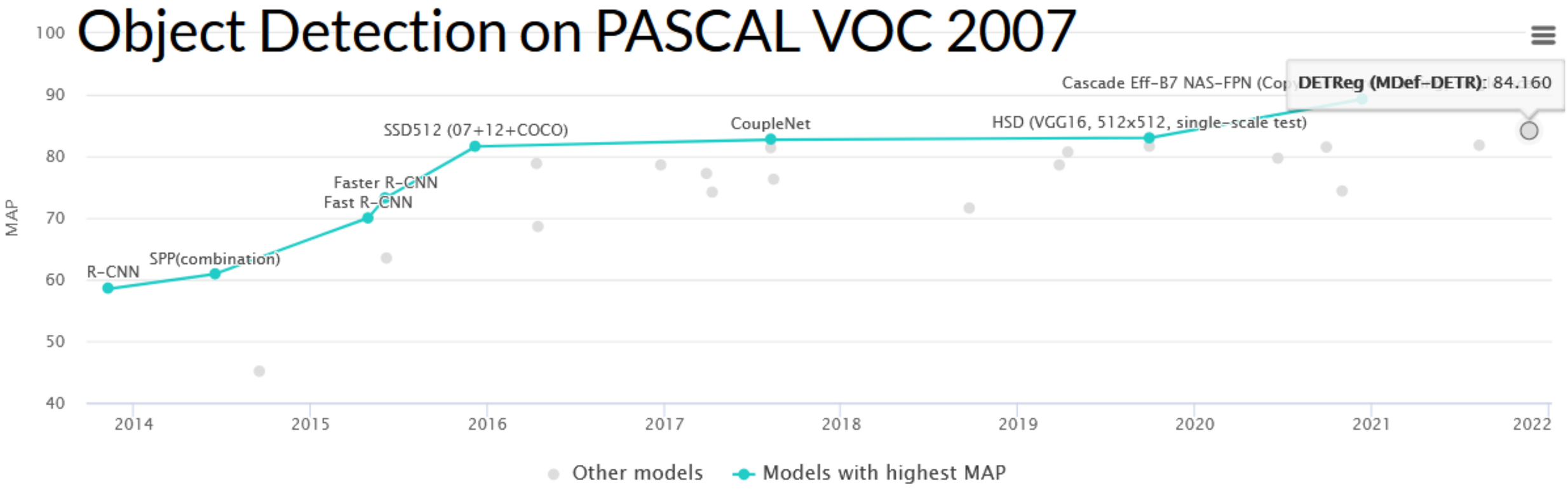
Figure 2. We use a simple copy and paste method to create new images for training instance segmentation models. We apply random scale jittering on two random training images and then randomly select a subset of instances from one image to paste onto the other image.

Model	FLOPs	# Params	AP_{val}	$AP_{test\text{-}dev}$	Mask AP_{val}	Mask $AP_{test\text{-}dev}$
SpineNet-190 (1536) [11]	2076B	176.2M	52.2	52.5	46.1	46.3
DetectoRS ResNeXt-101-64x4d [43]	—	—	—	55.7†	—	48.5 †
SpineNet-190 (1280) [11]	1885B	164M	52.6	52.8	—	—
SpineNet-190 (1280) w/ self-training [71]	1885B	164M	54.2	54.3	—	—
EfficientDet-D7x (1536) [56]	410B	77M	54.4	55.1	—	—
YOLOv4-P7 (1536) [60]	—	—	—	55.8†	—	—
Cascade Eff-B7 NAS-FPN (1280)	1440B	185M	54.5	54.8	46.8	46.9
w/ Copy-Paste	1440B	185M	(+1.4) **55.9**	(+1.2) **56.0**	(+0.4) **47.2**	(+0.5) **47.4**
w/ self-training Copy-Paste	1440B	185M	(+2.5) **57.0**	(+2.5) **57.3**	(+2.1) **48.9**	(+2.2) **49.1**

Table 4. Comparison with the state-of-the-art models on COCO object detection and instance segmentation. Parentheses next to the model name denote the input image size. † indicates results are with test time augmentation.

Golnaz Ghiasi et al «Simple Copy-Paste is a Strong Data Augmentation Methodfor Instance Segmentation» // https://arxiv.org/pdf/2012.07177v1.pdf

SotA 2022

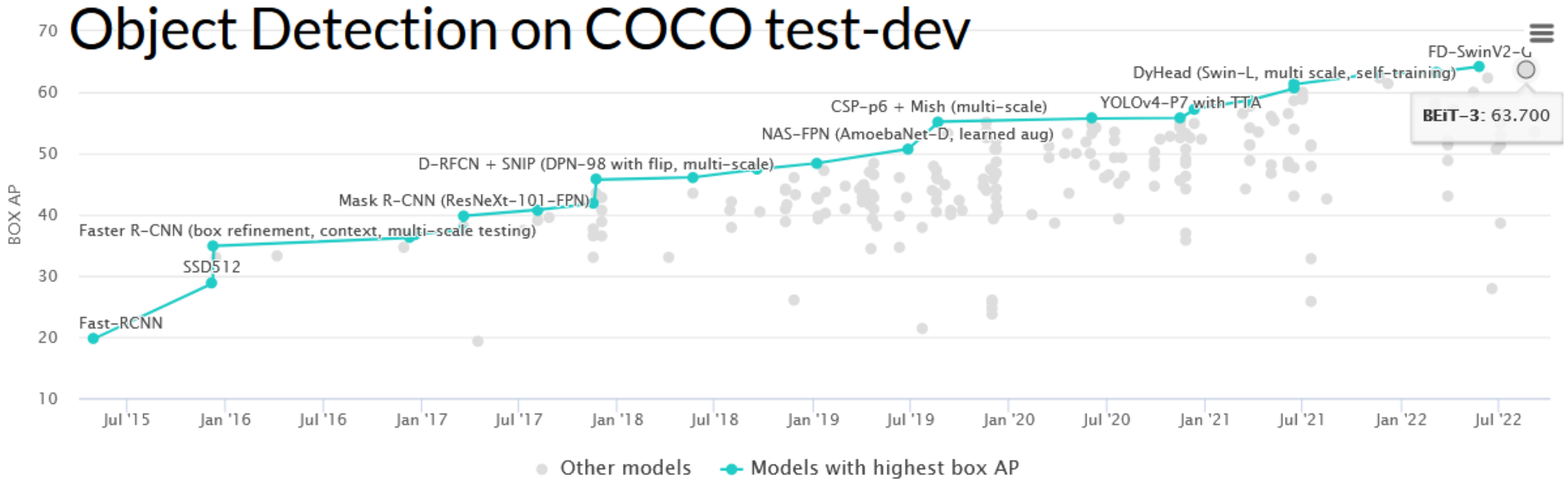


SotA – Copy-Paste

DETReg ~ MAVL (здесь)

https://paperswithcode.com/sota/object-detection-on-pascal-voc-2007

SotA 2022



здесь тоже дополнительные приёмы, например самообучение

https://paperswithcode.com/sota/object-detection-on-coco

SotA 2021

TABLE III: Performance comparison of various object detectors on MS COCO and PASCAL VOC 2012 datasets at similar input image size.

Model	Year	Backbone	Size	$AP_{[0.5:0.95]}$	$AP_{0.5}$	FPS
R-CNN*	2014	AlexNet	224	-	58.50%	~0.02
SPP-Net*	2015	ZF-5	Variable	-	59.20%	~0.23
Fast R-CNN*	2015	VGG-16	Variable	-	65.70%	~0.43
Faster R-CNN*	2016	VGG-16	600	-	67.00%	5
R-FCN	2016	ResNet-101	600	31.50%	53.20%	~3
FPN	2017	ResNet-101	800	36.20%	59.10%	5
Mask R-CNN	2018	ResNeXt-101-FPN	800	39.80%	62.30%	5
DetectoRS	2020	ResNeXt-101	1333	53.30%	71.60%	~4
YOLO*	2015	(Modified) GoogLeNet	448	-	57.90%	45
SSD	2016	VGG-16	300	23.20%	41.20%	46
YOLOv2	2016	DarkNet-19	352	21.60%	44.00%	81
RetinaNet	2018	ResNet-101-FPN	400	31.90%	49.50%	12
YOLOv3	2018	DarkNet-53	320	28.20%	51.50%	45
CenterNet	2019	Hourglass-104	512	42.10%	61.10%	7.8
EfficientDet-D2	2020	Efficient-B2	768	43.00%	62.30%	41.7
YOLOv4	2020	CSPDarkNet-53	512	43.00%	64.90%	31
Swin-L	2021	HTC++	-	57.70%	-	-

[a]Models marked with * are compared on PASCAL VOC 2012, while others on MS COCO.Rows colored gray are real-time detectors (>30 FPS).

$$AP_{[0.5:0.95]} = \text{mean}(A_{0.5}, A_{0.55}, \dots, A_{0.95})$$

Тренды

- AutoML
- Lightweight detectors
- Weakly supervised/few shot detection
- Domain transfer
- 3D object detection
- Object detection in video

Напоминание

по обзору

Xiongwei Wu, et al. «Recent Advances in Deep Learning for Object Detection»

https://arxiv.org/abs/1908.03673

История детектирования объектов

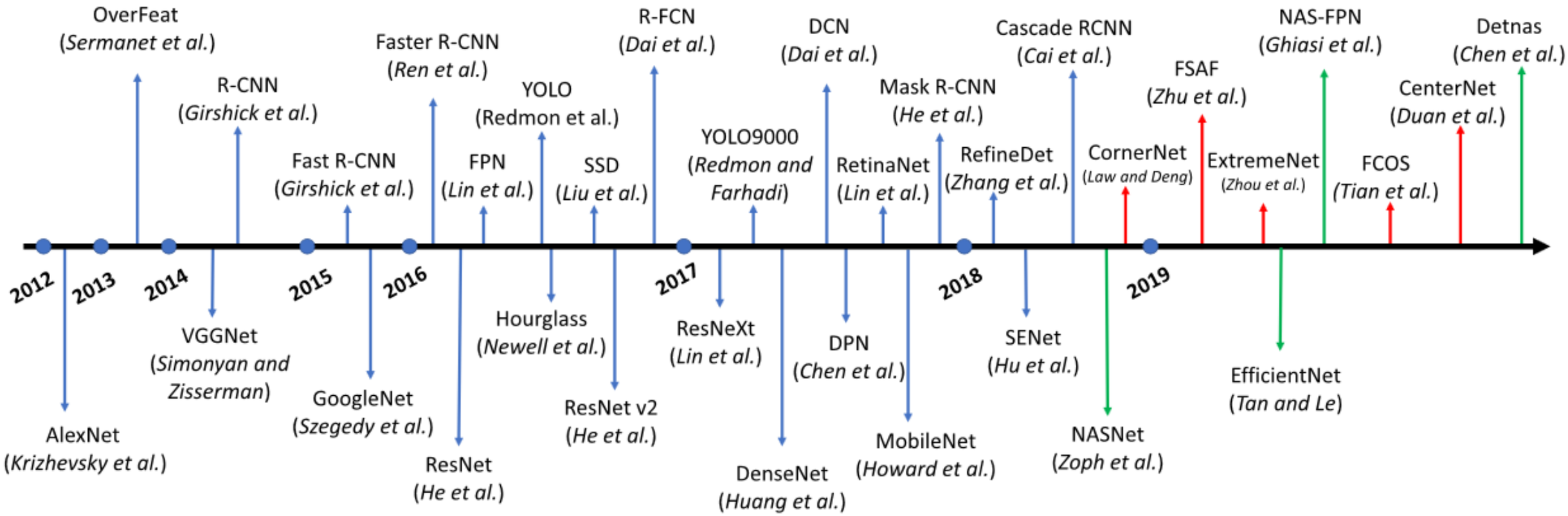
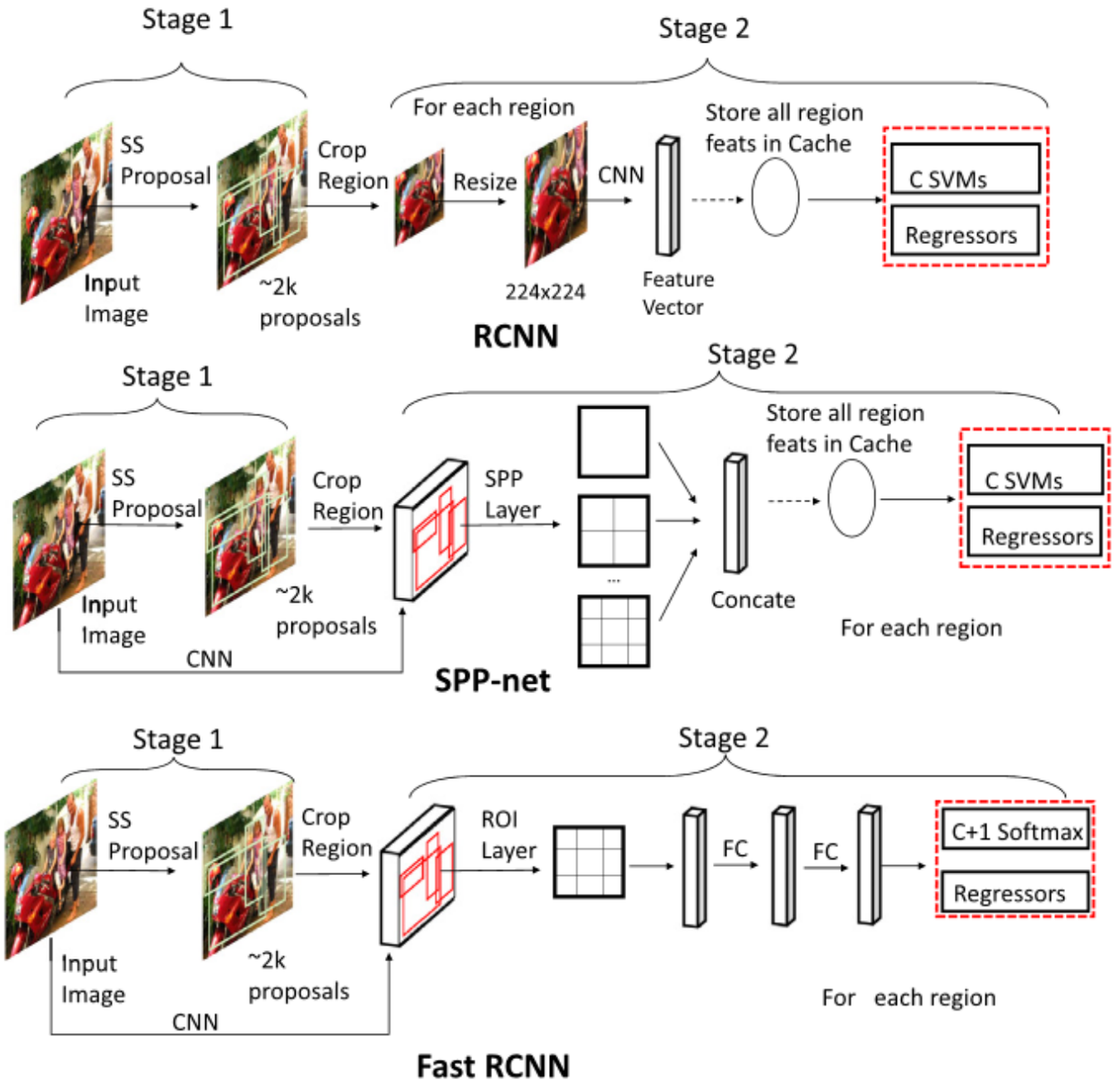


Fig. 2. Major milestone in object detection research based on deep convolution neural networks since 2012. The trend in the last year has been designing object detectors based on anchor-free (in red) and AutoML (in green) techniques, which are potentially two important research directions in the future. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

https://github.com/XiongweiWu/Awesome-Object-Detection



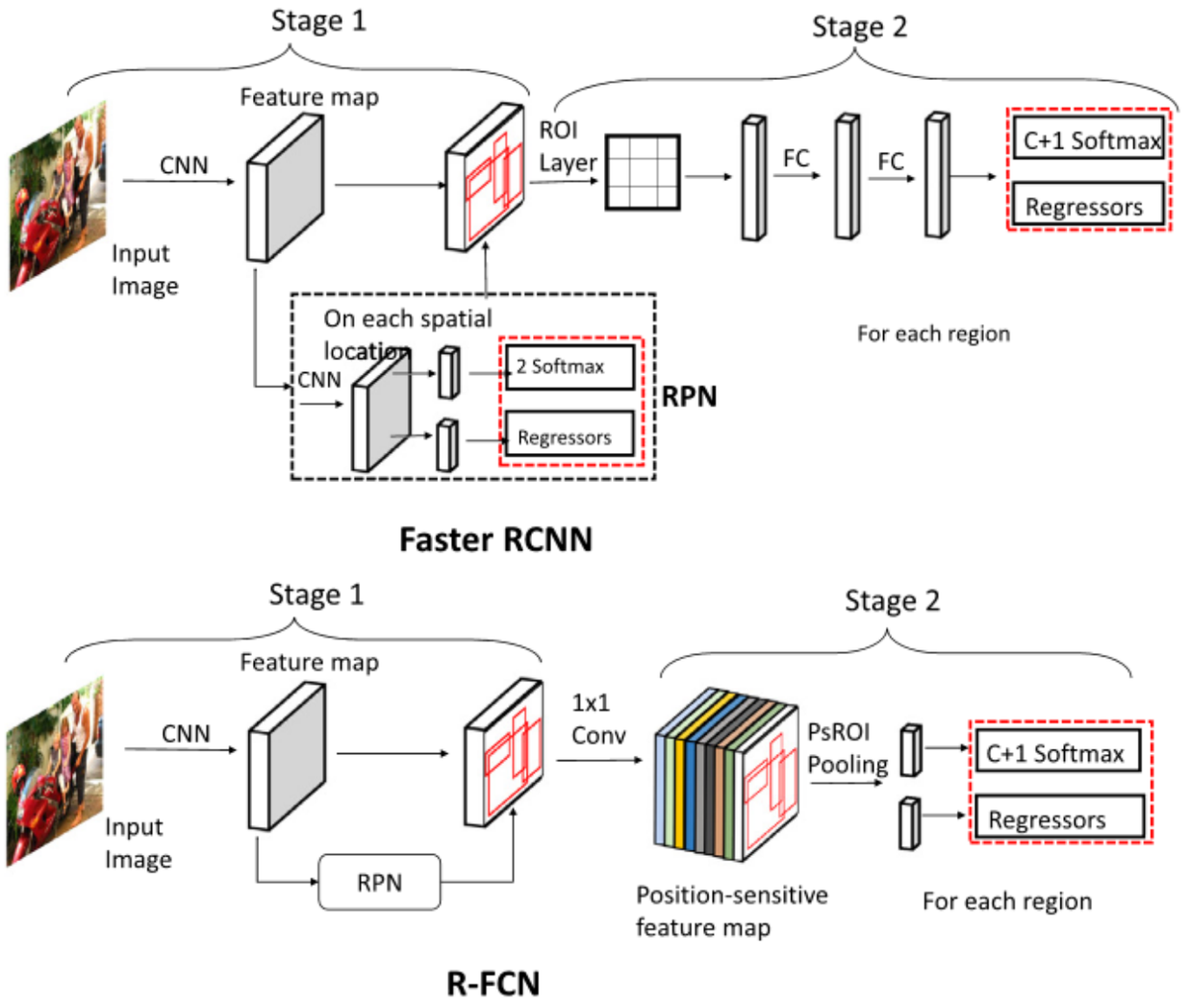


Fig. 4. Overview of different two-stage detection frameworks for generic object detection. Red dotted rectangles denote the outputs that define the loss functions. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

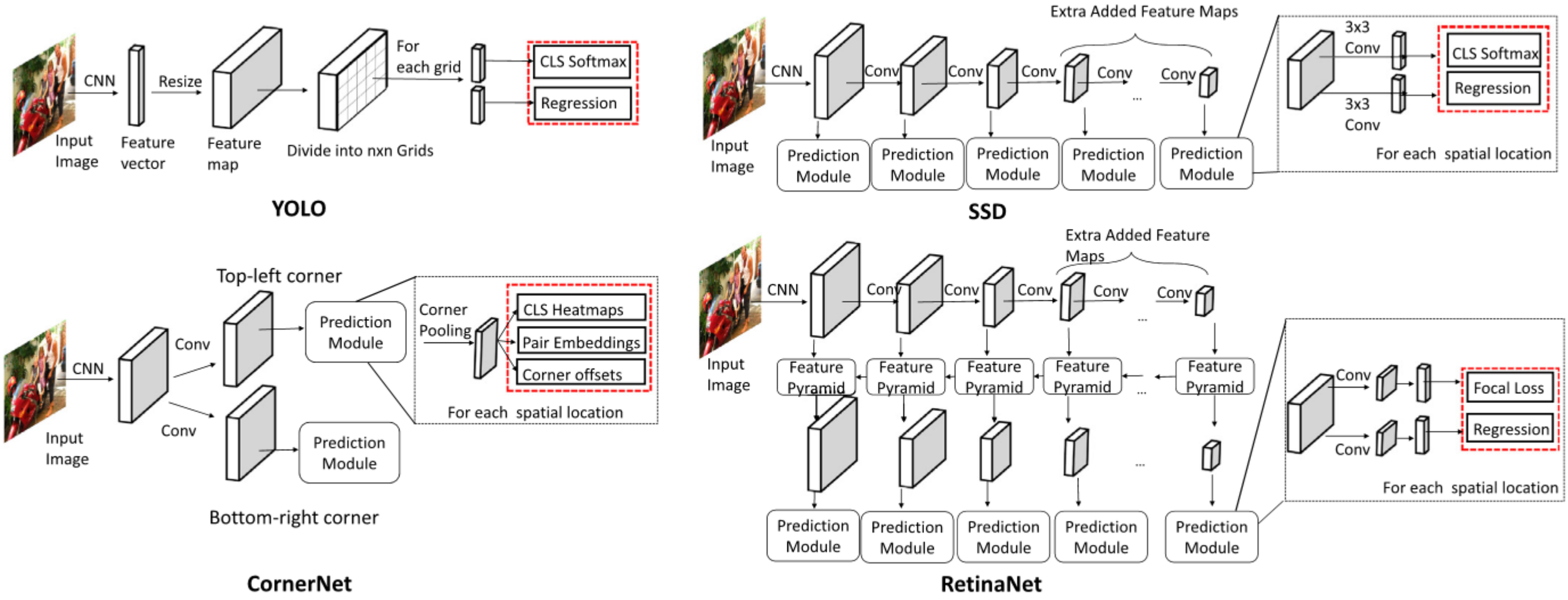


Fig. 5. Overview of different one-stage detection frameworks for generic object detection. Red rectangles denotes the outputs that define the objective functions.

Генерация признаков (архитектуры)

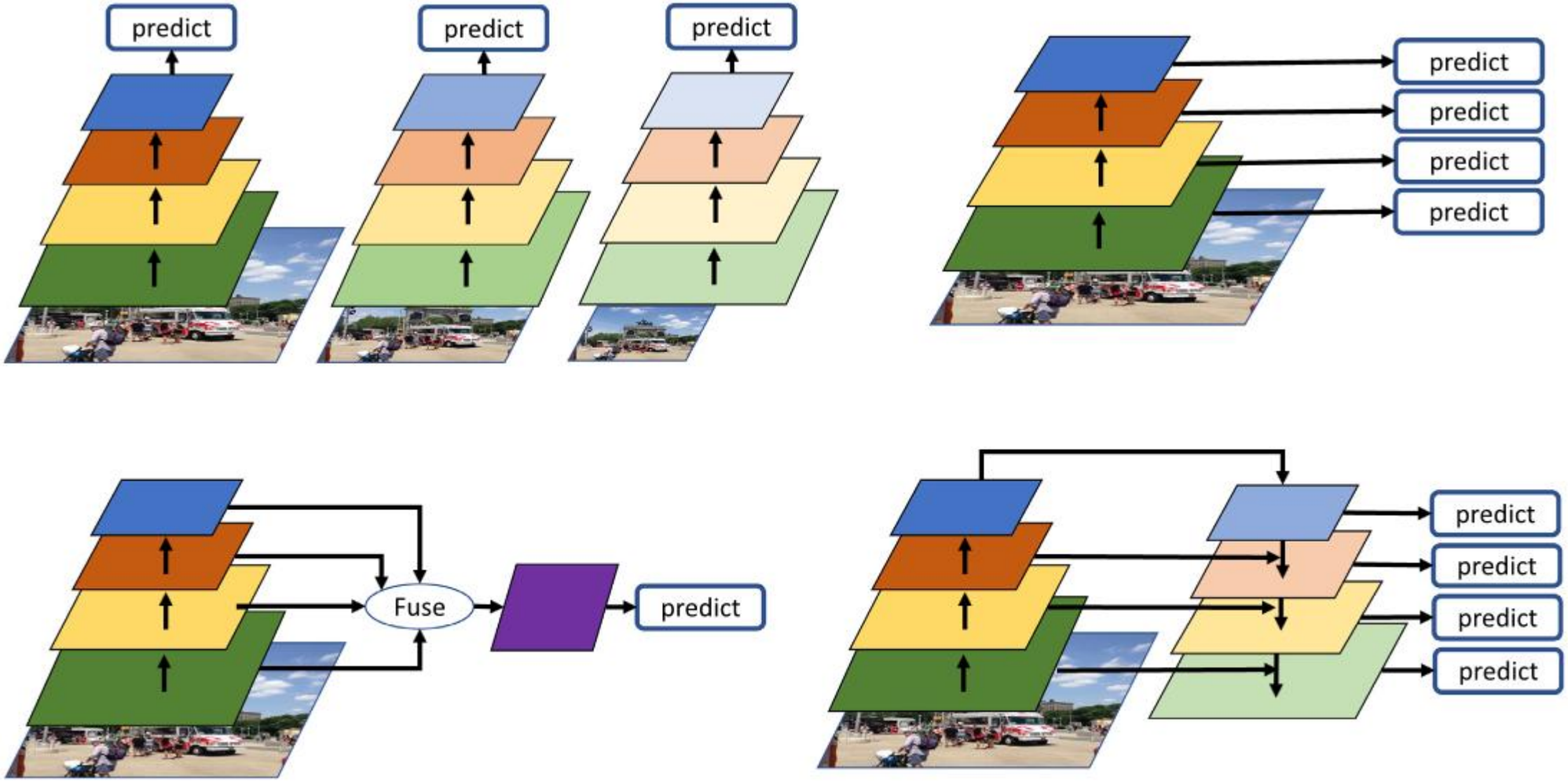


Fig. 7. Four paradigms for multi-scale feature learning. Top Left: *Image Pyramid*, which learns multiple detectors from different scale images; Top Right: *Prediction Pyramid*, which predicts on multiple feature maps; Bottom Left: *Integrated Features*, which predicts on single feature map generated from multiple features; Bottom Right: *Feature Pyramid* which combines the structure of *Prediction Pyramid* and *Integrated Features*.

Генерация признаков (архитектуры)

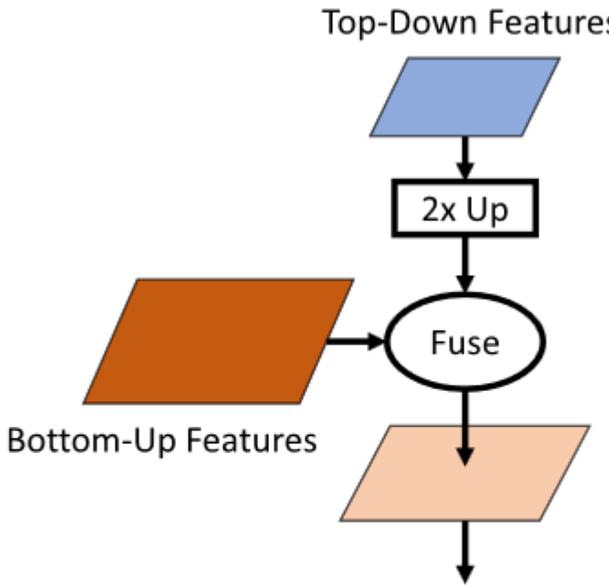


Fig. 8. General framework for feature combination. Top-down features are 2 times up-sampled and fuse with bottom-up features. The fuse methods can be element-wise sum, multiplication, concatenation and so on. Convolution and normalization layers can be inserted in to this general framework to enhance semantic information and reduce memory cost.

Подготовка признакового пространства (на базе любой сети из классификации)

можно предобучать

Смесь информации с разных разрешений

Определение параметров рамок

Общие проблемы в детекции

Дисбаланс классов

- искусственное формирование батчей
- Focal Loss

<table>
<tr><th colspan="4">Object Detection</th></tr>
<tr><th colspan="3">Detection Components</th><th>Learning Strategy</th><th>Applications & Benchmarks</th></tr>
<tr><th>Detection Settings</th><th>Detection Paradigms</th><th>Backbone Architecture</th><th>Training Stage</th><th>Applications</th></tr>
<tr><td rowspan="2">Bounding Box</td><td rowspan="2">Two-Stage Detectors</td><td rowspan="2">VGG16,ResNet,DenseNet</td><td>Data Augmentation</td><td rowspan="2">Face Detection</td></tr>
<tr><td>Imbalance Sampling</td></tr>
<tr><td rowspan="2">Pixel Mask</td><td rowspan="2">One-Stage Detectors</td><td rowspan="2">MobileNet, ResNeXt</td><td>Localization Refinement</td><td rowspan="2">Pedestrian Detection</td></tr>
<tr><td>Cascade Learning</td></tr>
<tr><td colspan="2" rowspan="2"></td><td rowspan="2">DetNet, Hourglass Net</td><td>Others</td><td rowspan="2">Others</td></tr>
<tr><td colspan="2">Proposal Generation</td><td>Feature Representation</td><td>Testing Stage</td><td>Public Benchmarks</td></tr>
<tr><td colspan="2">Traditional Computer Vision Methods</td><td>Multi-scale Feature Learning</td><td rowspan="2">Duplicate Removal</td><td rowspan="2">MSCOCO, Pascal VOC, Open Images</td></tr>
<tr><td colspan="2">Anchor-based Methods</td><td>Region Feature Encoding</td></tr>
<tr><td colspan="2">Keypoint-based Methods</td><td>Contextual Reasoning</td><td rowspan="2">Model Acceleration</td><td>FDDB, WIDER FACE</td></tr>
<tr><td colspan="2">Other Methods</td><td>Deformable Feature Learning</td><td rowspan="2">Others</td><td rowspan="2">KITTI, ETH, CityPersons</td></tr>
</table>

Fig. 3. Taxonomy of key methodologies in this survey. We categorize various contributions for deep learning based object detection into three major categories: Detection Components, Learning Strategies, Applications and Benchmarks. We review each of these categories in detail.

Итог

история перехода к полностью НС-решениям

локализация объектов не обязательно должна быть классоориентированной (class-specific)!

в одном регионе можно одновременно детектировать несколько объектов!

есть способ генерирования «якорей» – большого набора потенциальных регионов
есть способ обойтись без генерации регионов!

«Пирамидные сети» – способ формирования признакового пространства

Хорошая идея: каждая точка потенциальный представитель рамки (оцениваем её параметры)

Ссылки

Подборка материалов по детекции объектов

https://github.com/XiongweiWu/Awesome-Object-Detection

ещё одна…

https://github.com/XinZhangNLPR/awesome-anchor-free-object-detection

неплохой обзор от практика

https://machinethink.net/blog/object-detection/

самый последний (на 21.10.2021) обзор по детектированию объектов

«A Survey of Modern Deep Learning based Object Detection Models»

https://arxiv.org/pdf/2104.11892v2.pdf