Towards High-Efficiency Object Detection

Strategies and Techniques

Zihao Xiong

July 15, 2024

GrokCV

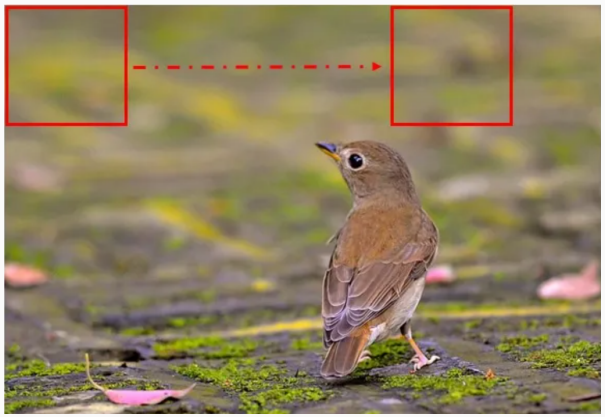
Table of contents

From Hand-Crafted Feature to End-to-End Feature Learning

Pre-Deep Learning Era

Traditional: Sliding Window Approach

- Hand-crafted features: SIFT, HOG



Selective Search

motivation: **enumeration $\Rightarrow$ candidate regions**

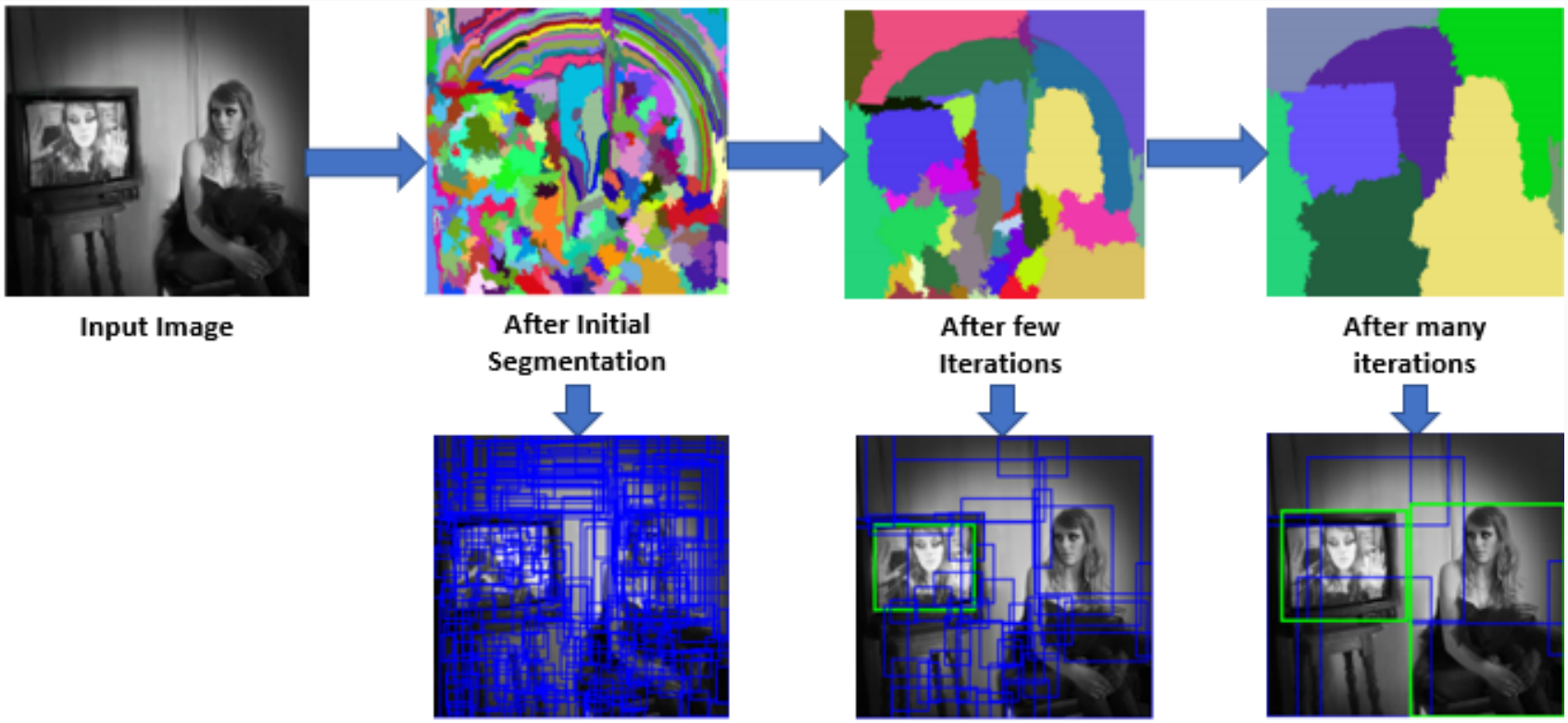


Fig. 2: Candidate region generation

Selective Search for Object Recognition

Selective Search combined with SIFT and more refined spatial pyramid partitioning for feature extraction

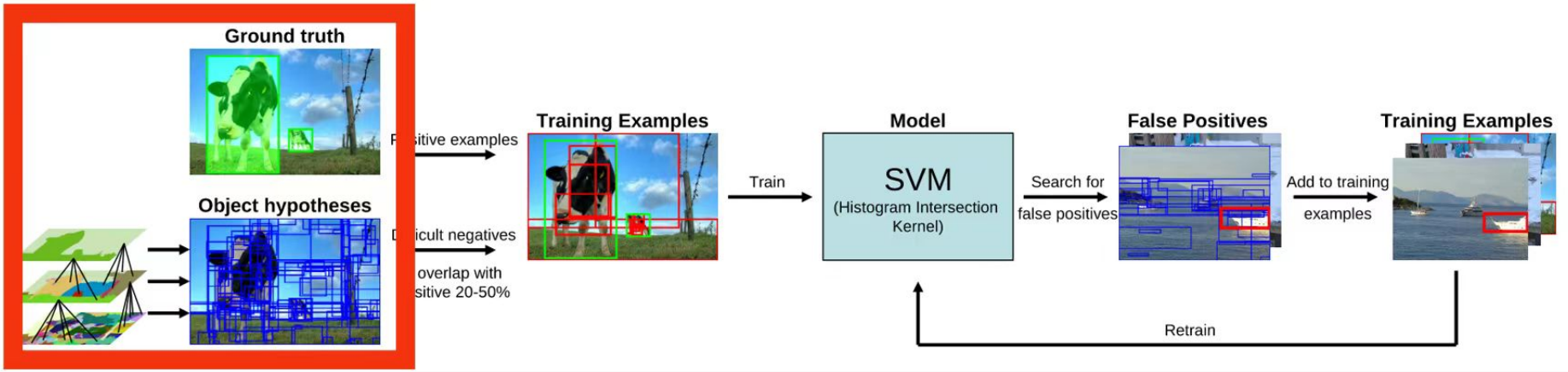


Fig. 3: Selective Search for Object Recognition

Deep Learning Era

Deep Feature: R-CNN

- 1. Candidate region generation
- 2. Feature extraction
- 3. Category judgment
- 4. Position refinement

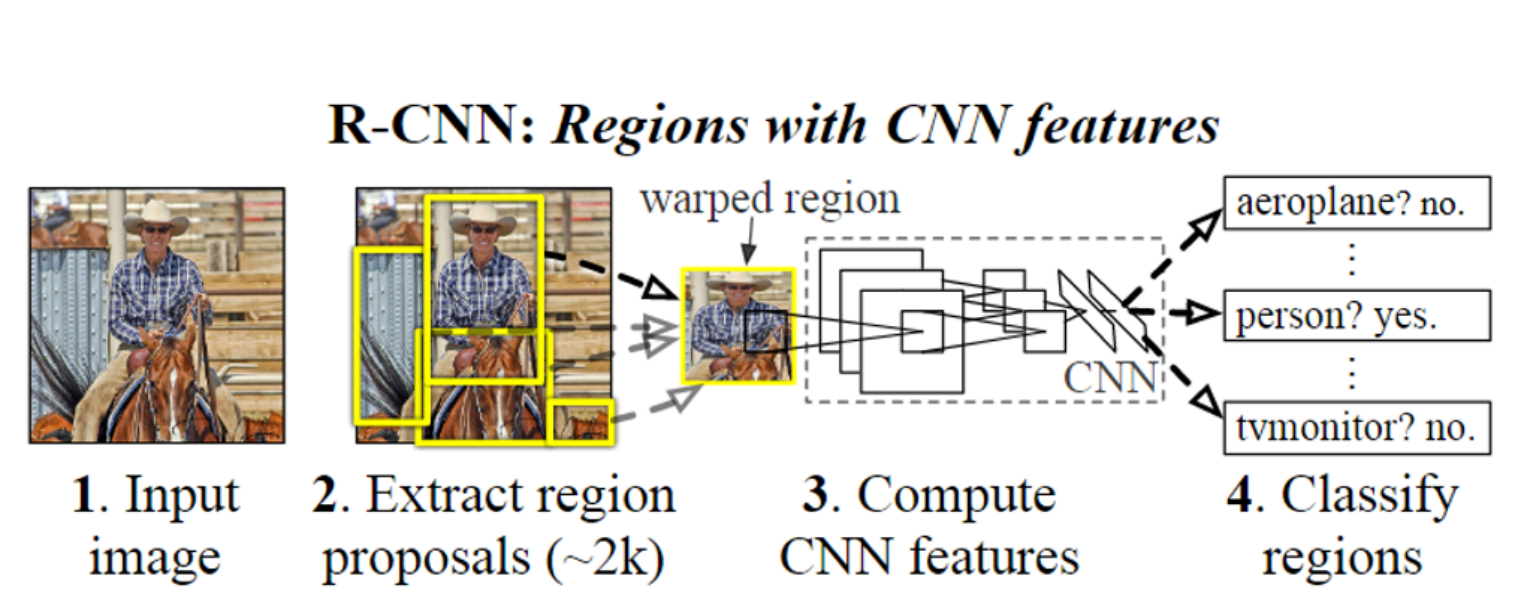


Fig. 4: R-CNN

Bounding Box Regression

Bounding Box Regression Parameters:
$[d_x, d_y, d_w, d_h]$

$$\hat{G}_x = P_w d_x(P) + P_x$$
$$\hat{G}_y = P_h d_y(P) + P_y$$
$$\hat{G}_w = P_w \exp(d_w(P))$$
$$\hat{G}_h = P_h \exp(d_h(P))$$

R-CNN Disadvantages:

1. Low computational efficiency
2. High storage overhead

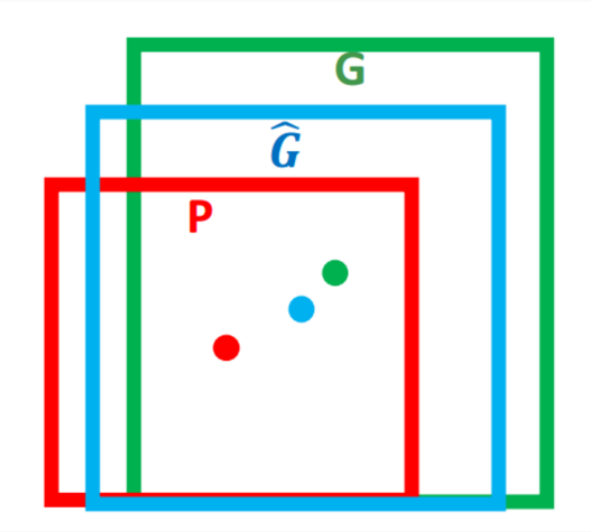


Fig. 5: Bounding Box Regression

- $\boldsymbol{G}$: Ground Truth
- $\hat{\boldsymbol{G}}$: Regression
- $\boldsymbol{P}$: Region Proposal

Share Computation: Fast R-CNN

Compared to R-CNN, the **improvements** of Fast R-CNN include:

1. **Single feature extraction**
2. **Remove SVM**

Disadvantages:

The generation of candidate regions (SS) is **too time-consuming**

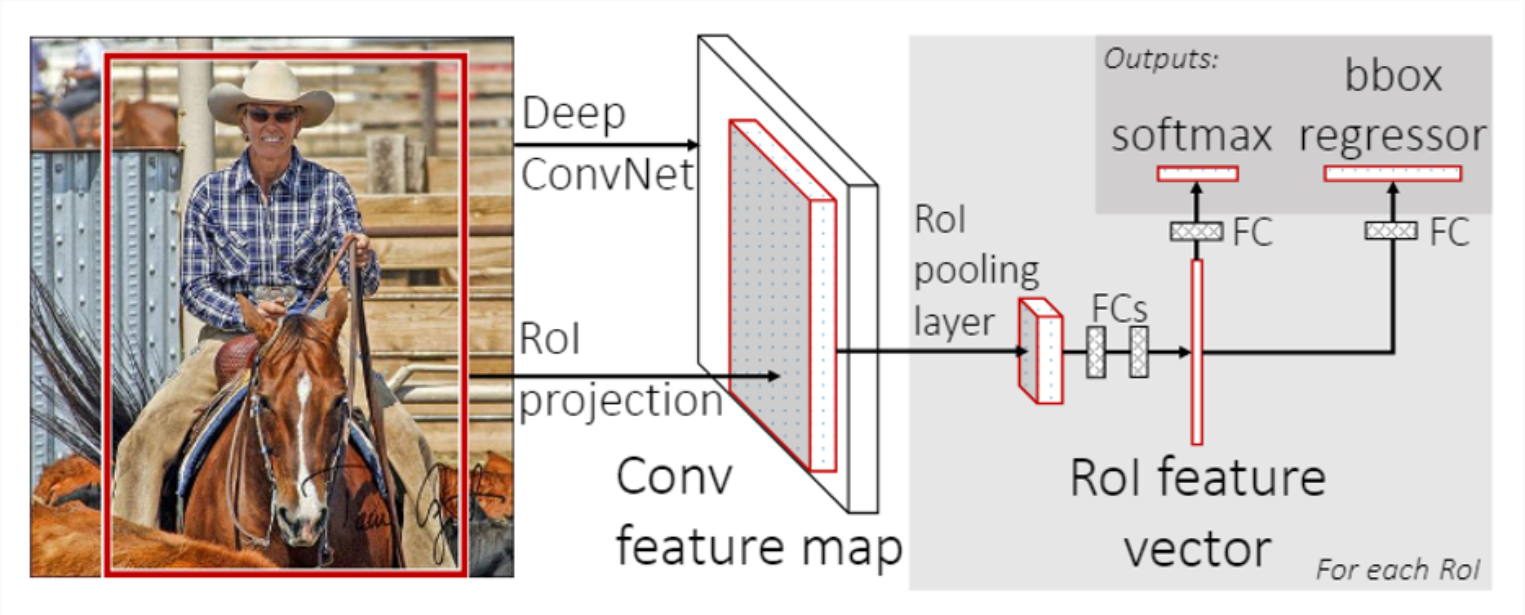


Fig. 6: Fast R-CNN

RoI pooling

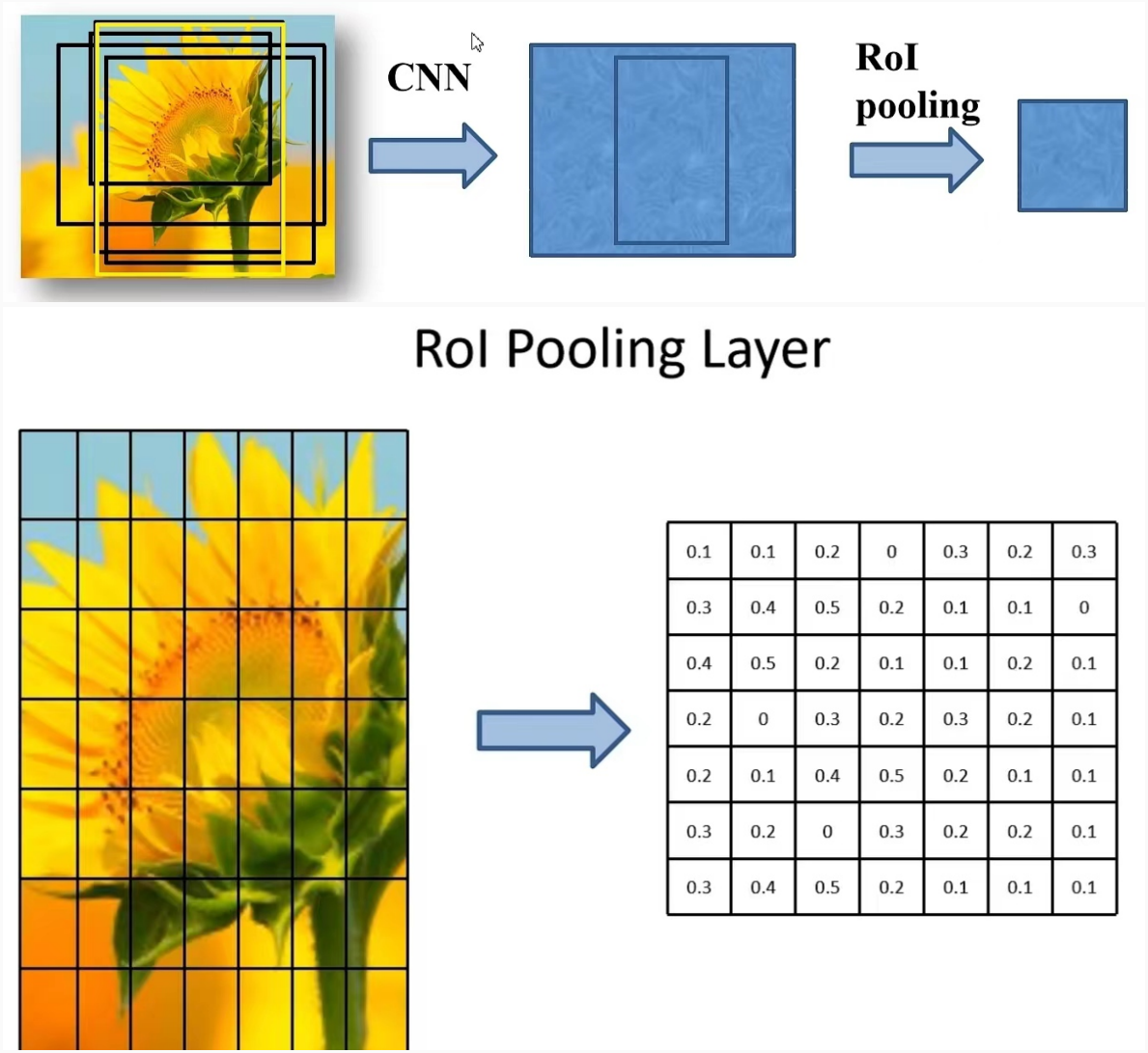


Fig. 7: RoI pooling

End-to-End Detection: Faster R-CNN

RPN (Region Proposal Network) is a convolutional neural network used to generate target candidate boxes. Compared to Selective Search, RPN can more effectively integrate with convolutional neural networks, enabling **end-to-end** training and inference.

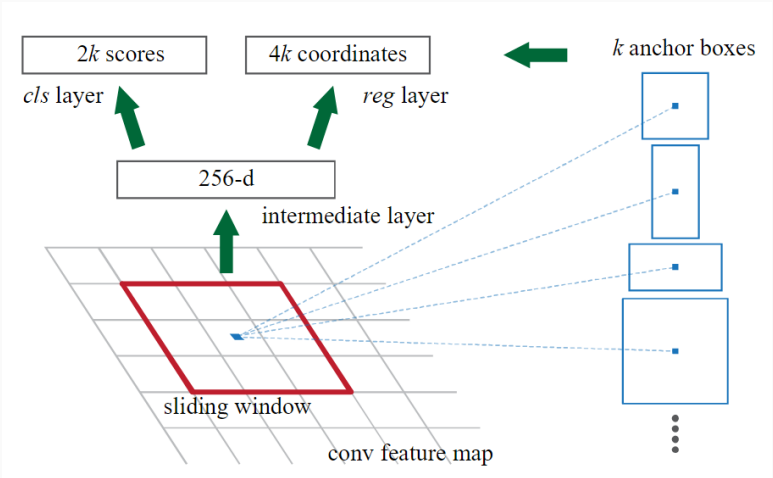


Fig. 8: RPN

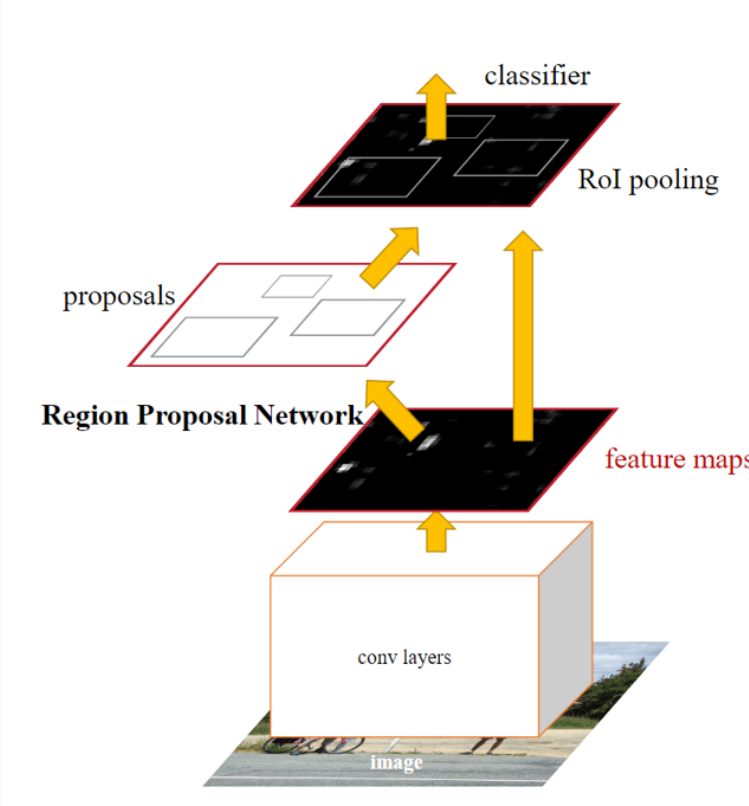
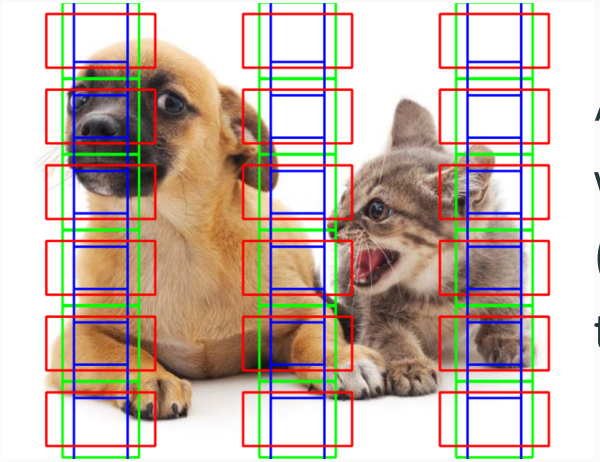


Fig. 9: Faster R-CNN

Revisiting Anchor Box Mechanism

Anchor Box Mechanism



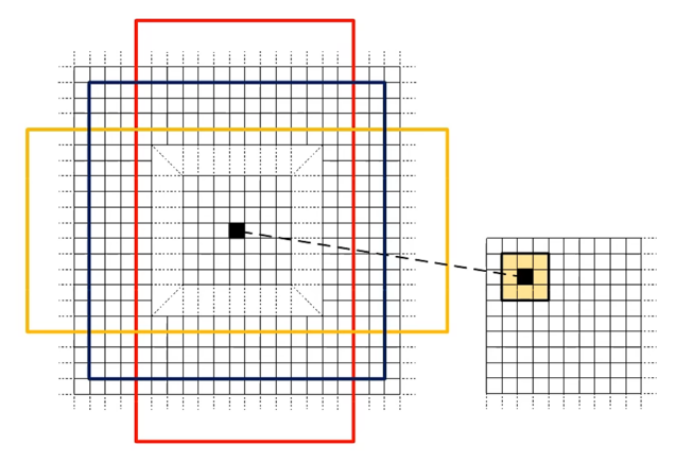
What are anchor boxes?

Anchor boxes are a set of **pre-defined** borders with different **aspect ratios** and **scales** (**actually, anchors are sliding windows on the shared feature maps**)

Why propose anchor box?

The objects that need to be detected may have many different shapes and sizes, and they may appear anywhere in the image.

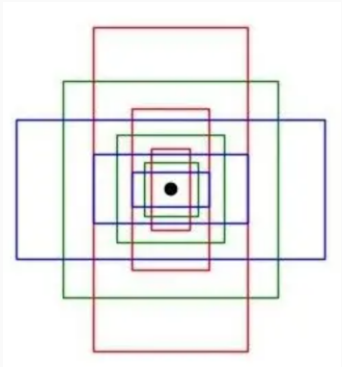
Anchor Box Mechanism



How are anchor boxes generated?

Assuming there is one scale and three aspect ratios here $(1:1, 1:2, 2:1)$

Map each point of the feature map to the original image and generate a set of anchor boxes at each mapping center

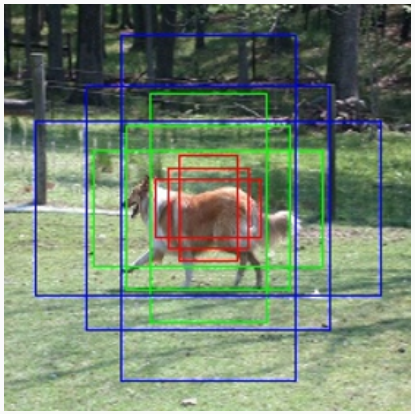


In Faster R-CNN

Using three scales $(128^2, 256^2, 512^2)$ and three aspect ratios (1:1, 1:2, 2:1)

Anchor Box Mechanism

The purpose of anchor boxes



Representing each **feature point** with two labels

1. the **category** of the object contained
2. The **offset** of the true bounding box

Match target box with anchor box:

Faster R-CNN:

- $\mathrm{IoU} \geq 0.7$, **positive samples**
- $\mathrm{IoU} < 0.3$, **negative samples**
- $\mathrm{IoU} \in [0.3, 0.7)$, **ignore**

RetinaNet:

- $\mathrm{IoU} \geq 0.5$, **positive samples**
- $\mathrm{IoU} < 0.4$, **negative samples**
- $\mathrm{IoU} \in [0.4, 0.5)$, **ignore**

From Two-Stage Detector to One-Stage Detector

Motivation

Why did we shift from two-stage detector to one-stage detector?

- Simplifying the model
- Increasing the speed

RetinaNet

RetinaNet

Focal Loss for Dense Object Detection

Tsung-Yi Lin Priya Goyal Ross Girshick Kaiming He Piotr Dollár
Facebook AI Research (FAIR)

- RetinaNet is a **classic anchor based object detection algorithm** and the first one stage network to **surpass two stages**
- **The extreme foreground background class imbalance** encountered during the training process of dense detectors is the main reason why the accuracy of one stage is lower than that of two stage detectors.
- Designed **RetinaNet network**, combined with **Focal Loss**, to significantly improve the accuracy of one stage detector.

RetinaNet network structure

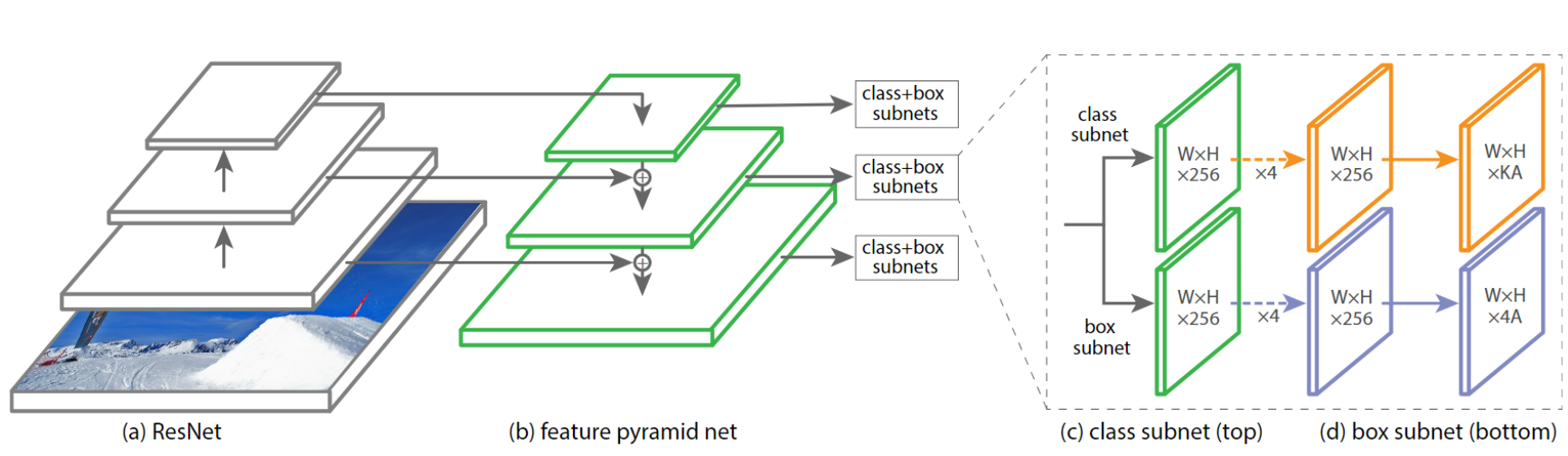


Fig. 10: RetinaNet network structure

RetinaNet

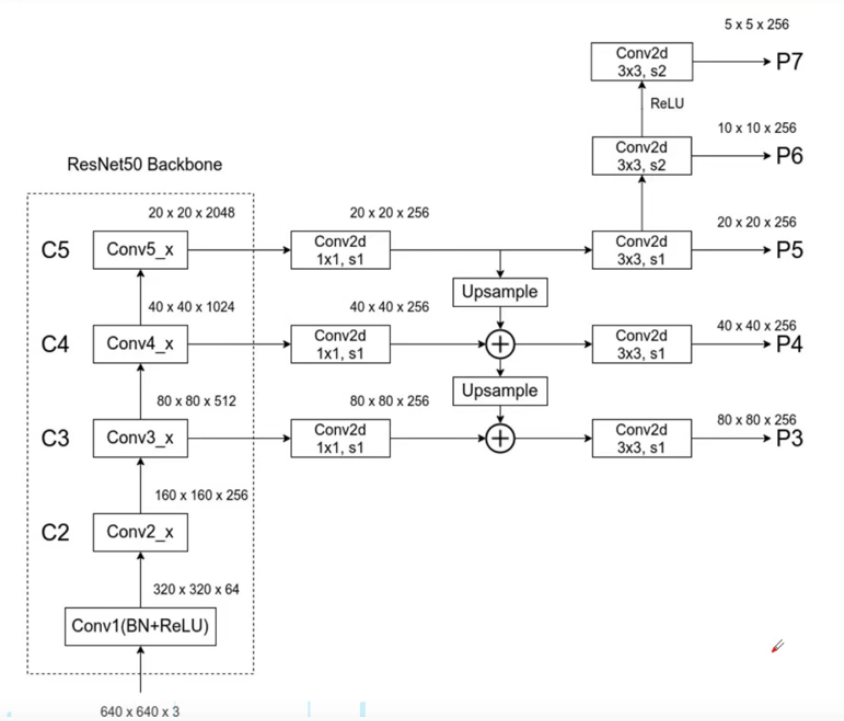
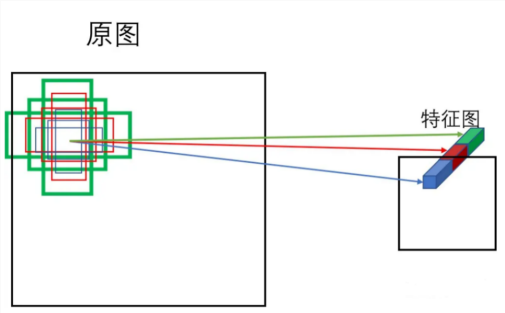


Fig. 11: RetinaNet



P_3-P_7 sride of feature map for: **[8, 16, 32, 64, 128]**

basesize set to: $[32^2, 64^2, 128^2, 256^2, 512^2]$

ratios [1:2, 1:1, 2:1] and scale [2^0, $2^{\frac{1}{3}}$, $2^{\frac{2}{3}}$] to form nine types of anchor boxes

RetinaNet

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [16]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [20]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [17]	Inception-ResNet-v2 [34]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [32]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [27]	DarkNet-19 [27]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [22, 9]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [9]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet (ours)	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet (ours)	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2

From Dense Detector to Sparse Detector

A Brief Introduction to SPConv

Sparse Convolution

Essentially, it is achieved by establishing a **hash table** that stores **values** at specific locations, **weights** in the convolutional kernel, and **calculation results**

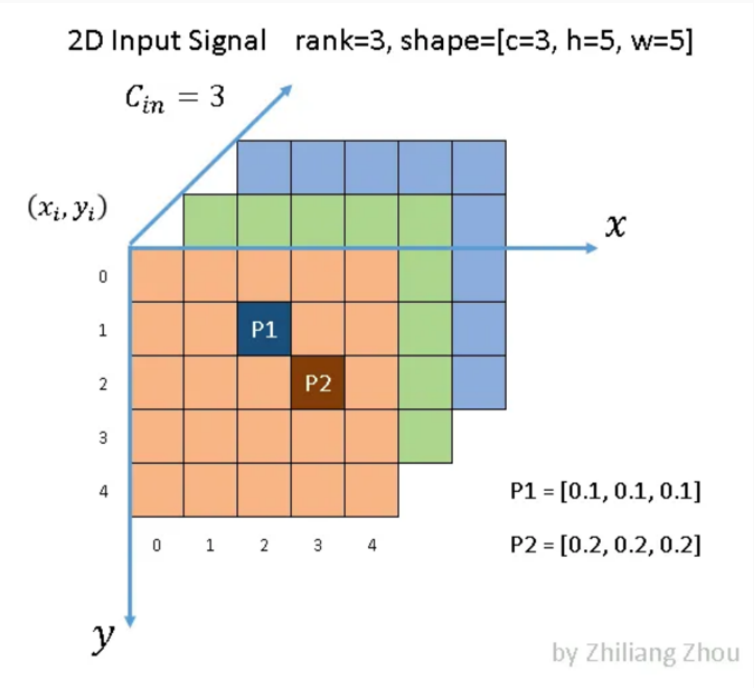


Fig. 12: Sparse Convolution

Calculation process:

1. Establish a hash table
2. Build Rulebook
3. Perform sparse calculations

Sparse Convolution

Two common output definitions:

- **regular output definition**
- **submanifold output definition**

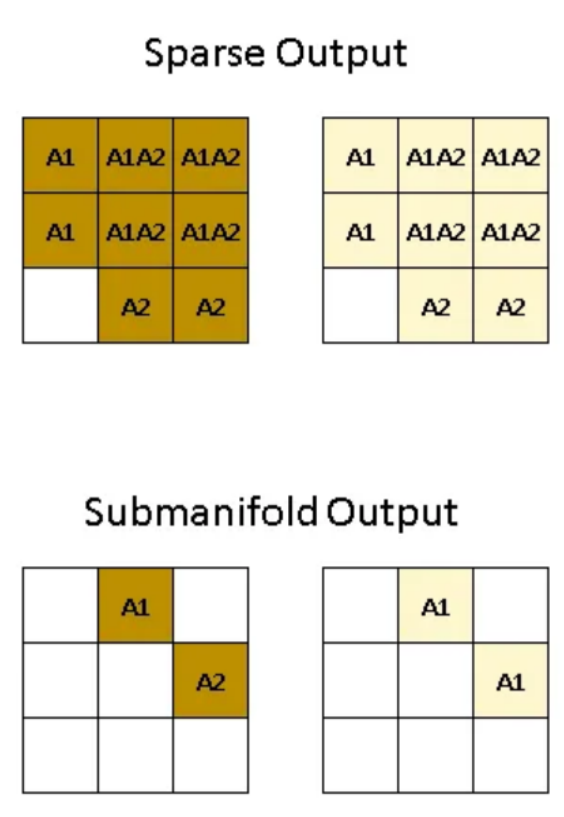


Fig. 13: Sparse Convolution

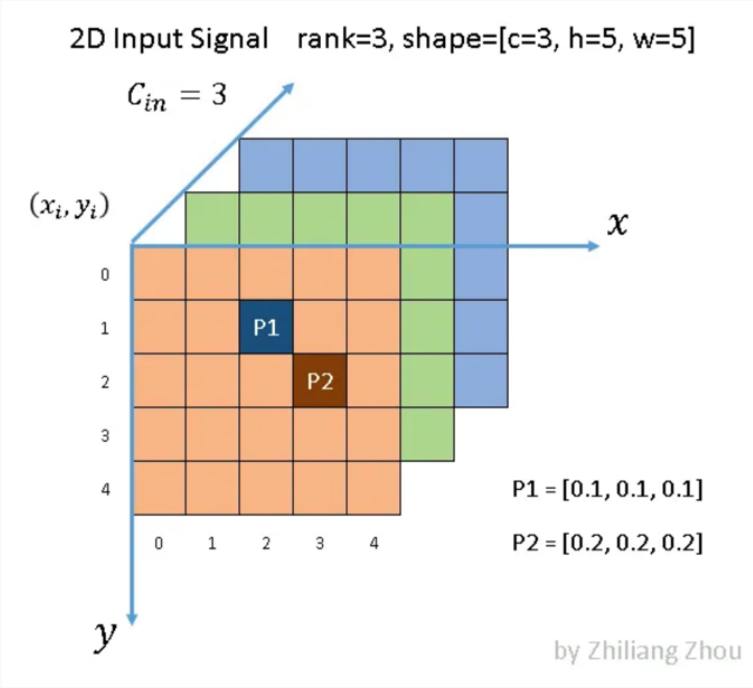


Fig. 14: Sparse Convolution

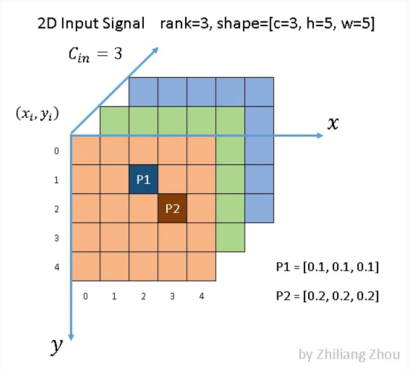
Sparse Convolution

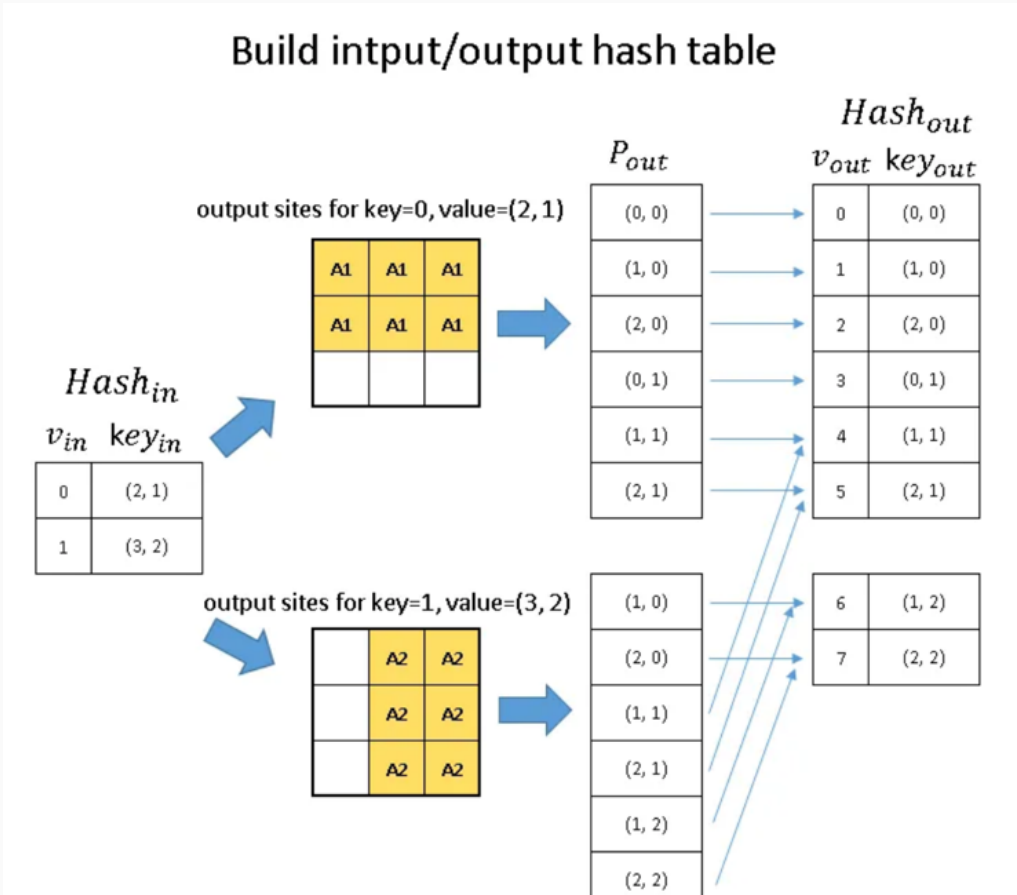
Firstly, establish an input hash table $hash_{in}$:

- V_{in}: **input serial number**
- Key_{in}: **Value at the position of the input graph**

Then establish a hash table $hash_{out}$:

- V_{in}: **input serial number**
- Key_{in}: **Value at the position of the input graph**

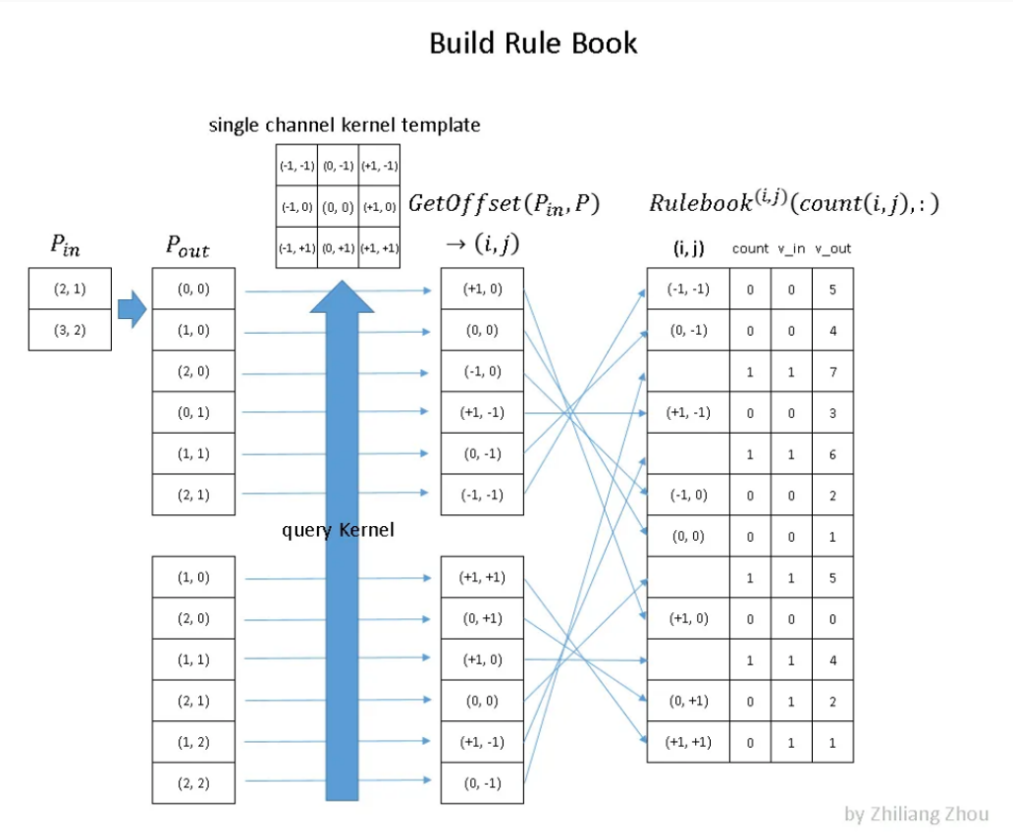




Sparse Convolution

Establish Rulebook:

1. From P_{out} to GetOffset()
2. From GetOffset() to Rulebook

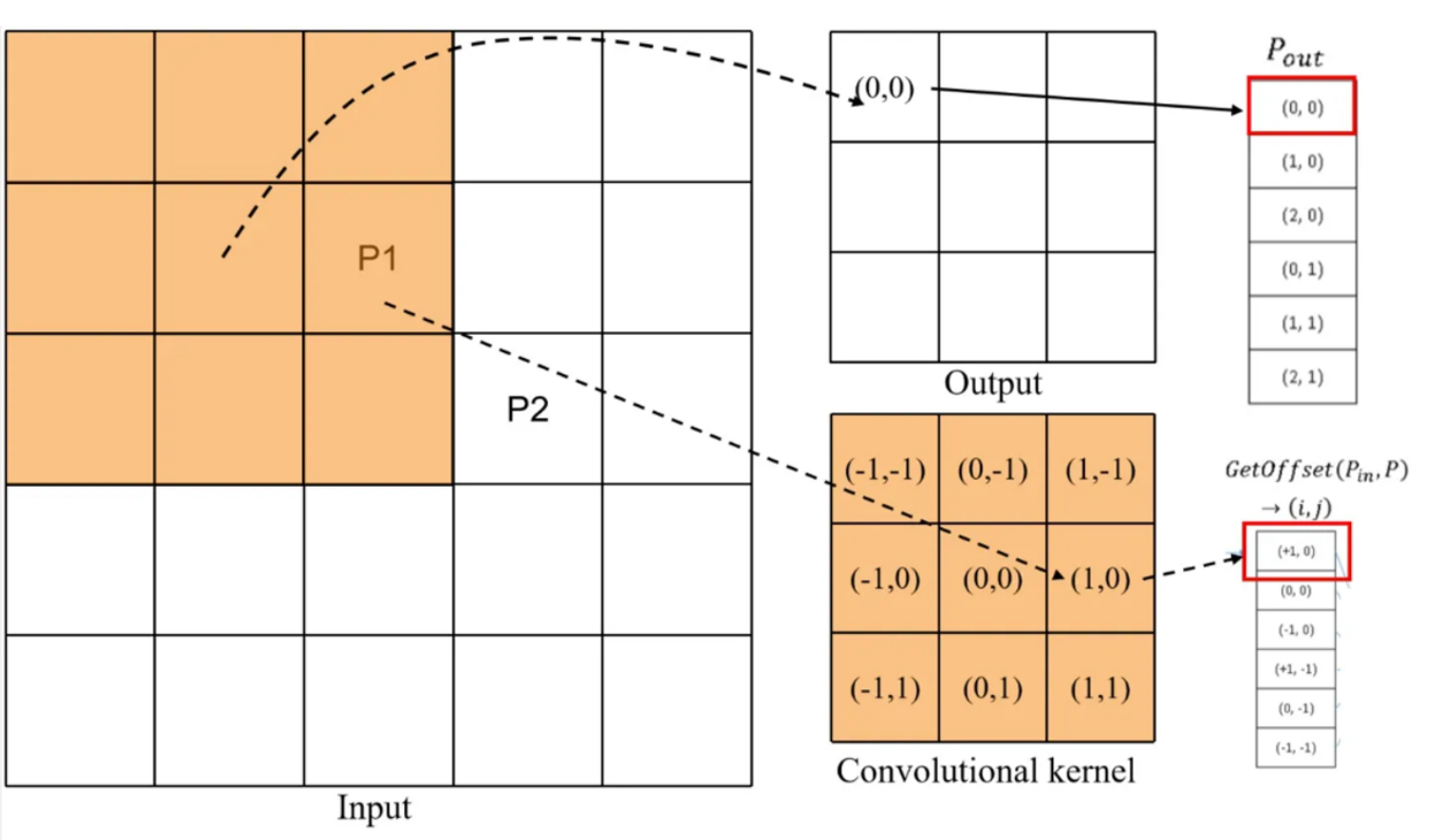


- (i, j): **Weight position**
- V_{in}: **Input sequence number of pixels**
- V_{out}: **The output sequence number corresponding to the convolution result**

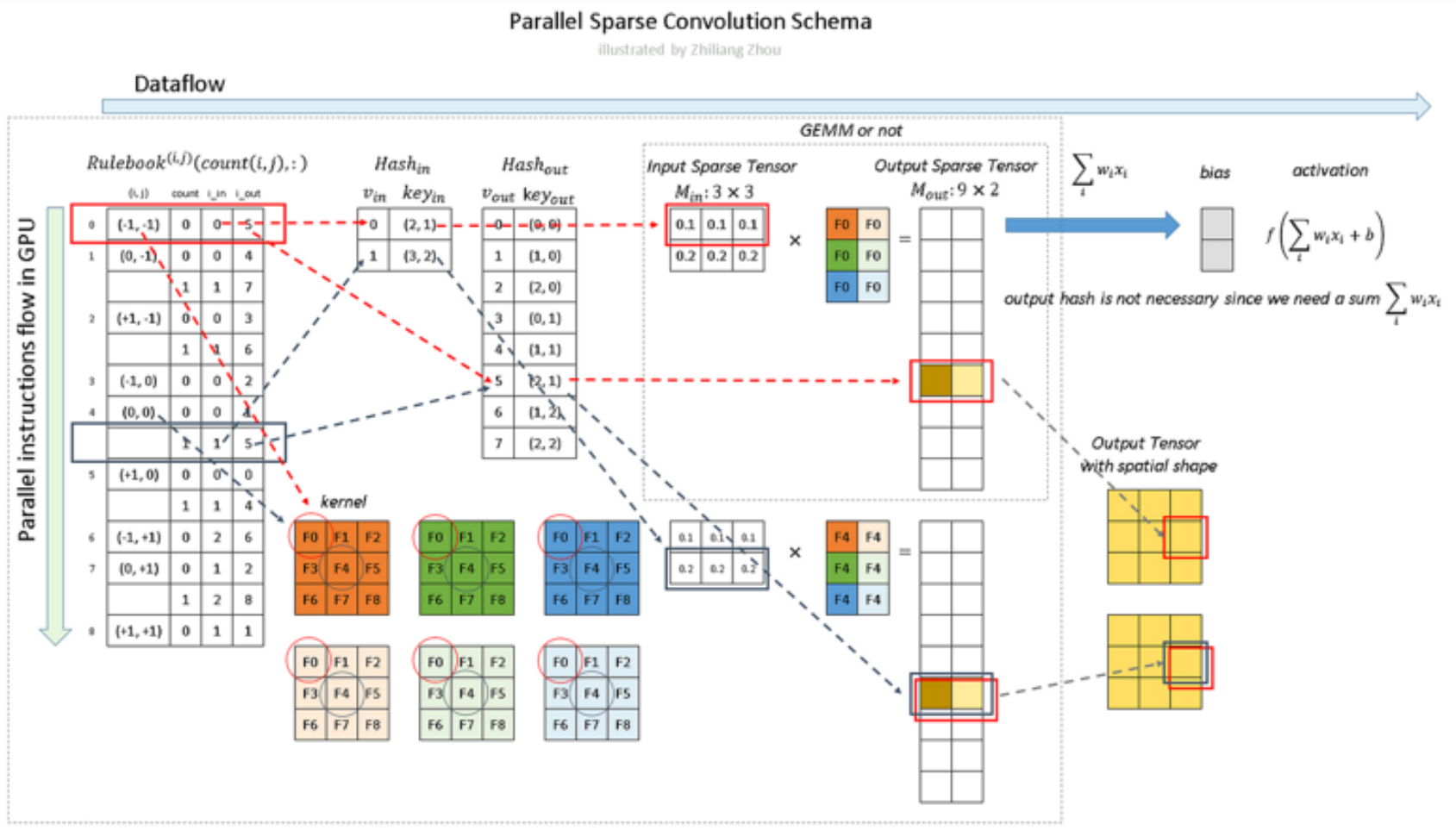
Sparse Convolution

From GetOffset() to Rulebook

GetOffset() is used to find a certain position in the output that needs to be calculated using the weight in the convolutional kernel



Sparse Convolution



QueryDet

Performance and efficiency issues in detecting small objects

The performance degradation in small object detection is mainly caused by three factors:

AP_S	AP_M	AP_L
15.6	38.7	50.9
18.2	39.0	48.2
13.5	38.1	52.0
16.2	39.8	**52.1**
5.0	22.4	35.5
10.2	34.5	49.8
13.0	35.4	51.1
21.8	42.7	50.2
24.1	**44.2**	51.2

1. **downsampling operations**
2. **Scale mismatch**
3. **Deviation of IoU**

Motivation

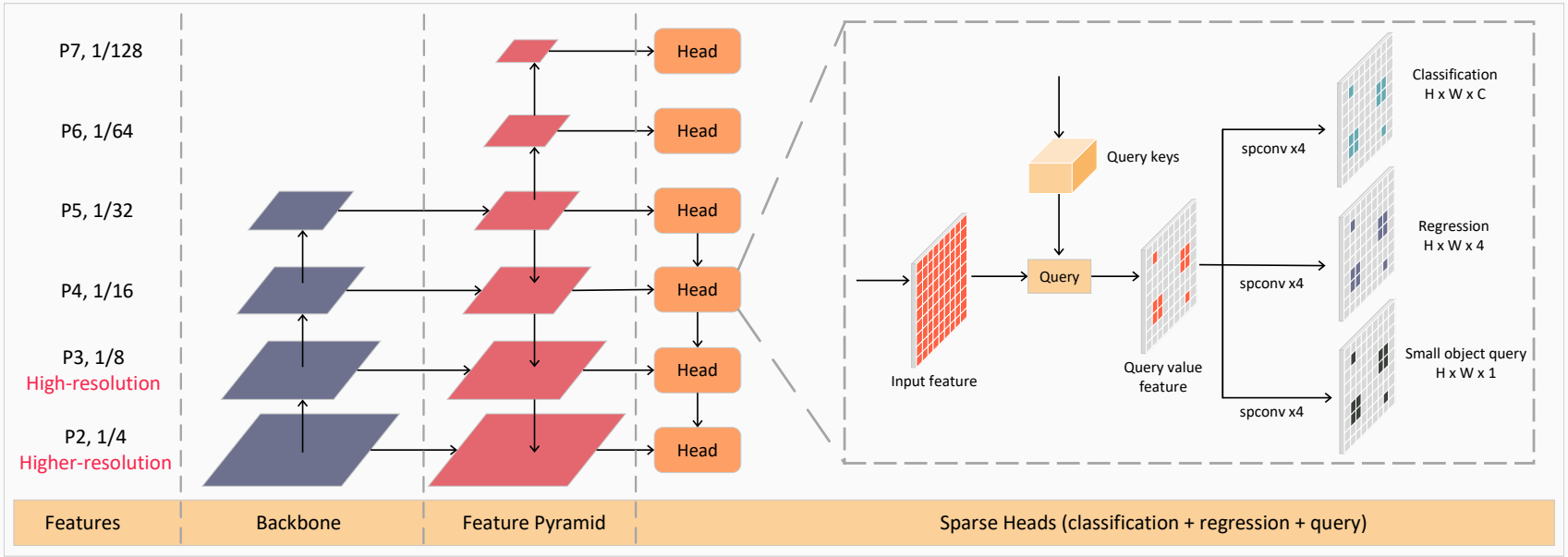
Motivation comes from two key observations:

1. **The calculation of low-level features is highly redundant**
2. In the FPN structure, it is possible to roughly determine the existence and corresponding regions of small objects with **high confidence.**

Network Architecture

QueryDet is generally a process from coarse to fine

QueryDet has added an additional query head in the detection head, which is parallel to the classification and registration heads, to generate small object seam locations.



Cascade Sparse Query (CSQ) Mechanism

Idea: **Recursively** predict the rough positions of small objects (queries) on low resolution feature maps and use them to **guide calculations in higher resolution feature maps.**

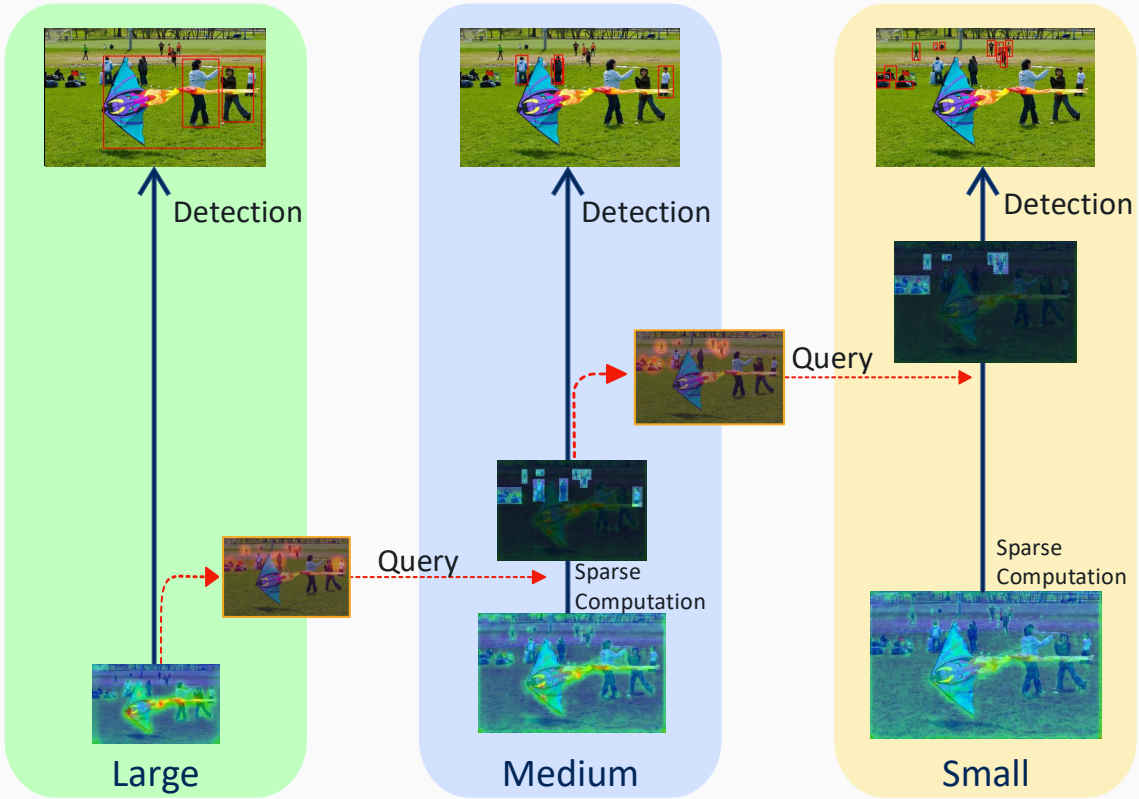


Fig. 15: CSQ

Label matching

During training,set the GT box for small object o on P_l to $b_l^0 = (x_l^0, y_l^o, w_l^o, h_l^o)$. Calculate the minimum distance map between each feature position (x, y) on P_l and all small ground truth centers (x_l^0, y_l^o):

$$D_l[x][y] = \min_o \left\{ \sqrt{(x - x_l^o)^2 + (y - y_l^o)^2} \right\}$$

The ground truth query map is defined as:

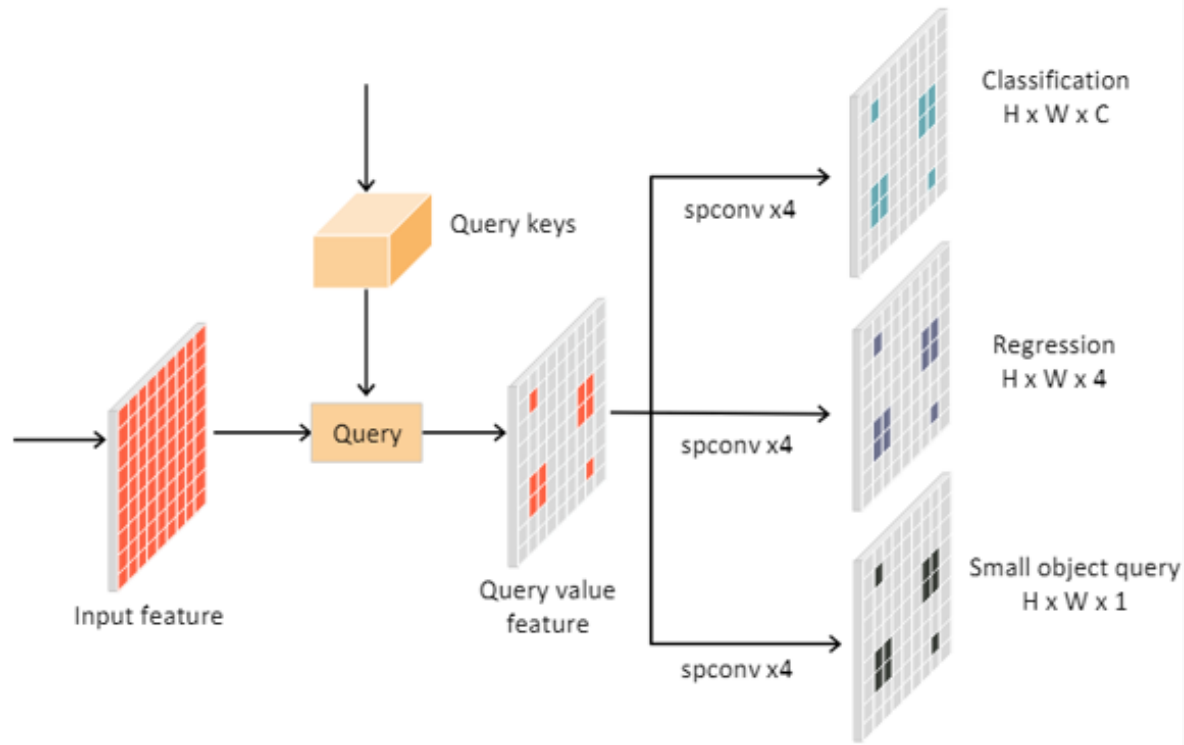
$$V_l^*[x][y] = \begin{cases} 1 & \text{if } D_l[x][y] < s_l \\ 0 & \text{if } D_l[x][y] \geq s_l \end{cases}$$

- S_l: Minimum anchor frame size for each layer of FPN.

Accelerating Inference by Sparse Query

During the inference process, we choose to predict a score greater than the threshold σ The location of is used as the query.
Assigned to the next layer P_{l-1}, q_l^o will be assigned to its nearest four points as the key position:

$$\left\{k_{l-1}^o\right\} = \left\{\left(2x_l^o + i, 2y_l^o + j\right), \forall i, j \in \{0, 1\}\right\}$$



Visualization: Accelerating Inference by Sparse Query

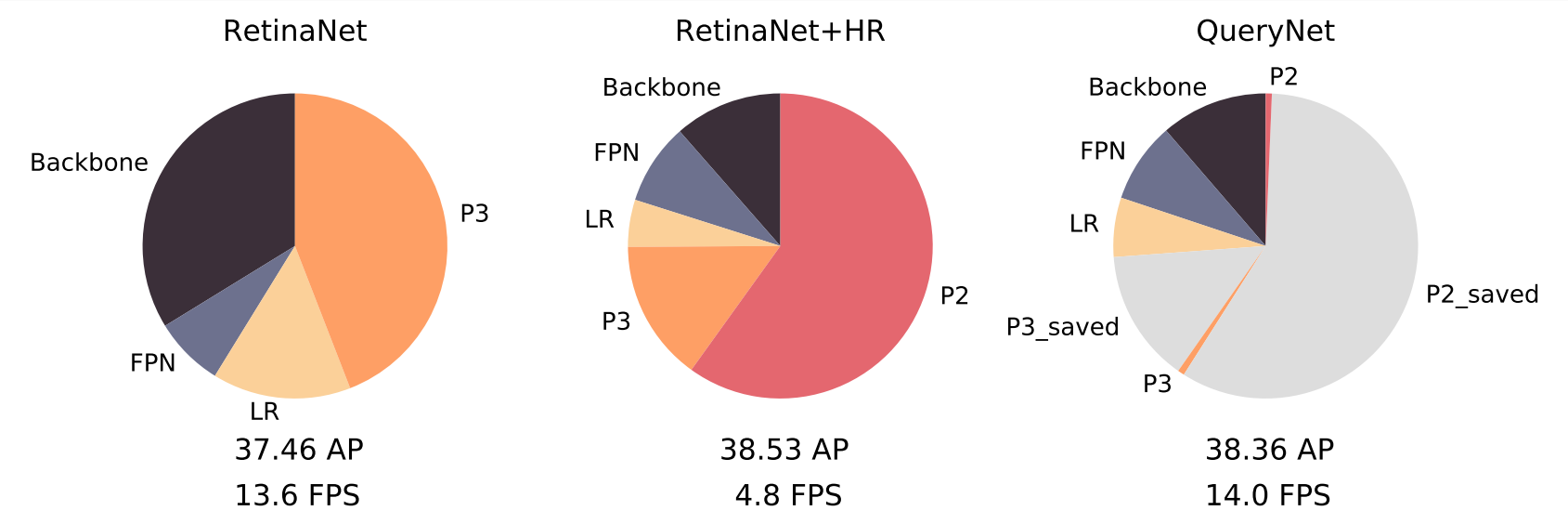
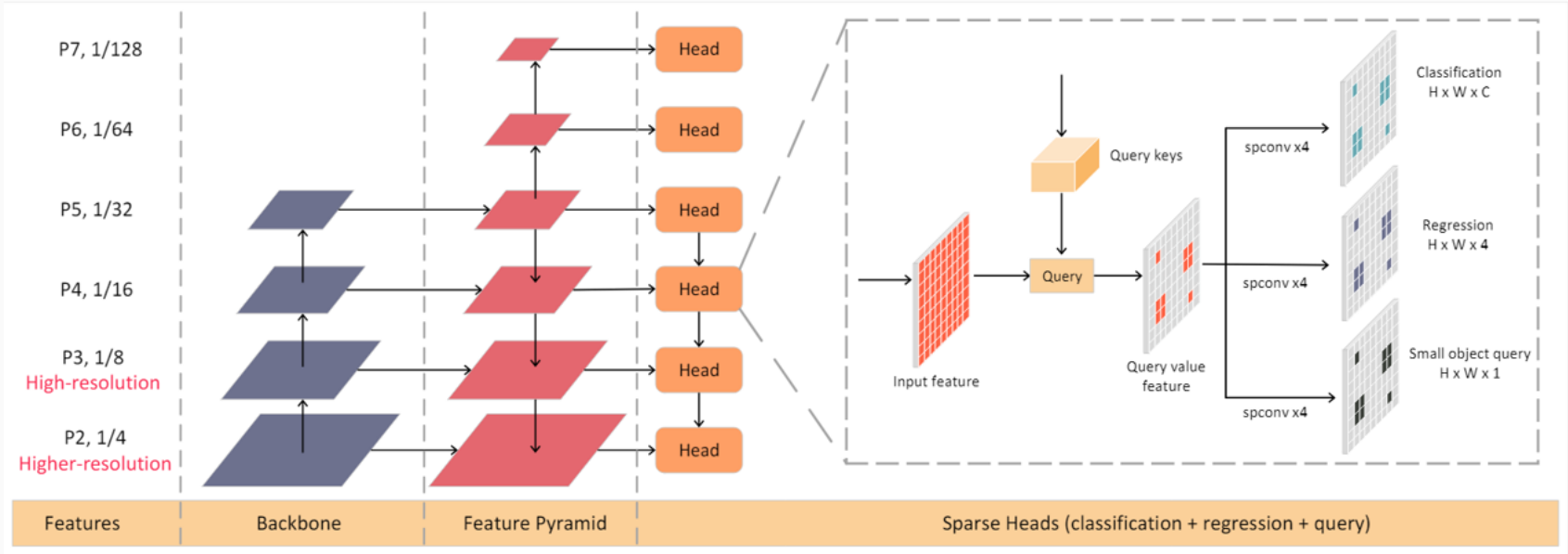


Fig. 17: When using the ResNet-50 backbone, the FLOP distribution of different modules is represented by LR representing low resolution P_4 to P_7.

Network Architecture

QueryDet is generally a process from coarse to fine

QueryDet has added an additional query head in the detection head, which is parallel to the classification and registration heads, to generate small object seam locations.



Ablation Study: Selection of starting layer

Start Layer	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	FPS
No Query	38.53	59.11	41.12	24.64	41.97	49.53	4.86
P_6	37.91	57.98	40.51	23.18	42.02	49.53	13.42
P_5	38.22	58.55	40.86	23.65	42.00	49.53	13.92
P_4	38.36	58.78	40.99	24.33	41.97	49.53	14.88
P_3	38.45	58.94	41.07	24.50	41.93	49.52	11.51

Two reasons why CSQ did not start from the lowest resolution layer:

- For low resolution features, normal convolution is very fast.
- Difficult to distinguish small objects on feature maps with very low resolution.

Ablation Study: Impact of CSQ

QueryDet: High resolution object detection optimized for small objects, while **improving detection performance and speed**

Method	CSQ	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	FPS
RetinaNet	-	37.46	56.90	39.94	22.64	41.48	48.04	13.60
RetinaNet (3x)	-	38.76	58.27	41.24	22.89	42.53	50.02	13.83
QueryDet	×	38.53	59.11	41.12	24.64	41.97	49.53	4.85
QueryDet	✓	38.36	58.78	40.99	24.33	41.97	49.53	14.88
QueryDet (3x)	×	39.47	59.93	42.11	25.24	42.37	51.12	4.89
QueryDet (3x)	✓	39.34	59.69	41.98	24.91	42.38	51.12	15.94

Table 1. Comparison of accuracy (AP) and speed (FPS) of our QueryDet and the baseline RetinaNet on COCO *mini-val* set.

Questions?

Thanks for listening!