名企 Project

AI for CV Group
2021

Week 1-4
Detection-3 Stages

Contents:

○. Introduction

○. Introduction

OA. Some Advices

OA1. Google

- **You have to know how to get access to Google without any excuse. That is a MUST.**

○. Introduction

OA. Some Advices

OA2. Coding (test)

- Coding Test
- Extremely Critical
- 200 / 300+
- 40min, 4 / 5, medium
- Strategy / Material
- Lintcode / Leetcode / 九章算法

○. Introduction

OA. Some Advices

OA3. Coding (test)

Lintcode 刷题（基础题部分+提高部分）

1. Beginning （4 + 3）

627	667
13	841
415	594
200	

2. Binary Search & Log(n) Algorithm （10 + 18）

458	462	457
585	459	141
460	235	617
447	254	586
428	28	160
159	14	63
140	414	437
75	61	183
74	38	
62	600	

Lintcode 刷题（基础题部分）

1. Beginning （4）

627
13
415
200

2. Binary Search & Log(n) Algorithm （10）

458
585
460
447
428
159
140
75
74
62

○. Introduction

OB. Project introduction

○. Introduction

OC. Course Schedule

Detection II 1-Stage	Algorithm Tricks I Data/Reg/Act/…
Detection II 1-Stage (continue)	Algorithm Tricks II Loss…
Detection II Code	Acceleration Quant/Slim
Detection III Anchor Free	Backup

○. Introduction

OD. Self Introduction

OD1. Experiences

- **CAS**
- **UFL**
- **Bay Area**
- **Alibaba**
- **Bitmain / Sensetime**
- **Startup**

○. Introduction

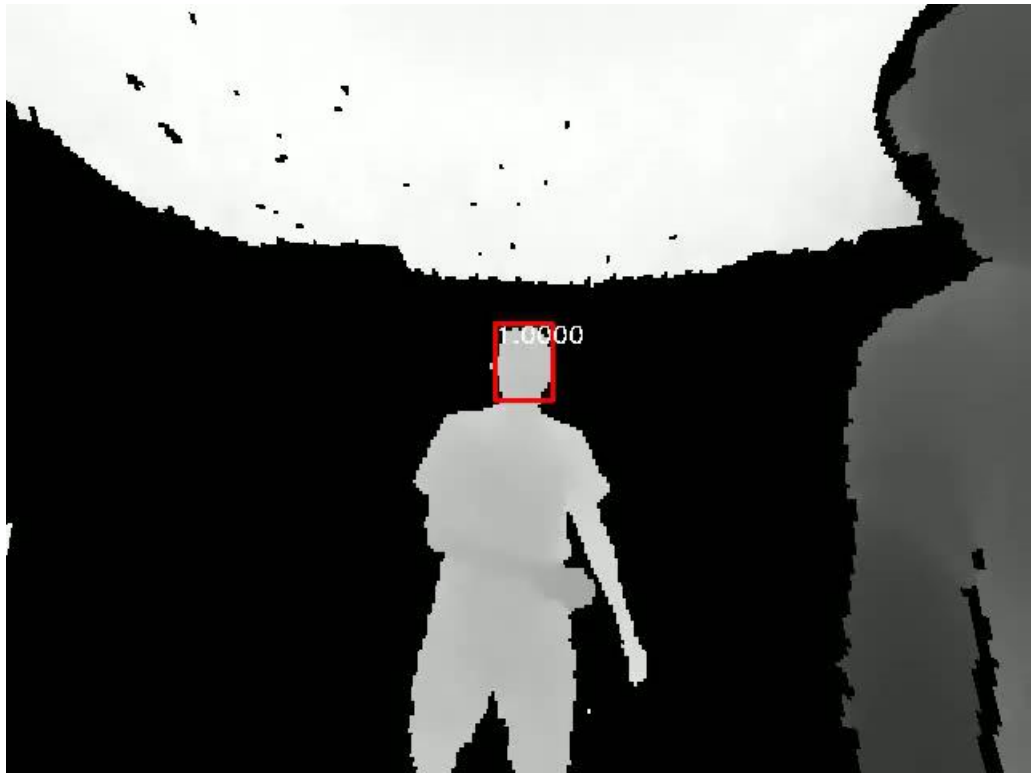
OD. Self Introduction

OD1. Demos

○. Introduction

OD. Self Introduction

OD2. Demos

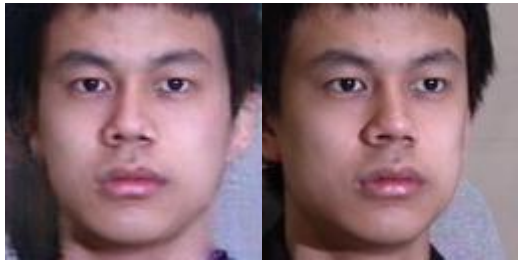


○. Introduction

OD. Self Introduction

OD3. Demos



试用期期间工资为 转正工资的 100% ，试用期结束之后税前 肆万元 人民币每月

Contents:

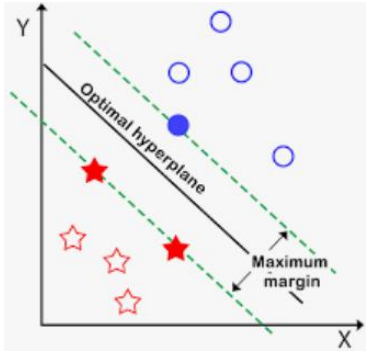
Contents:

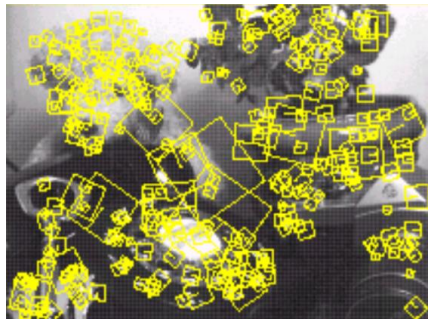
III. Anchor Free Methods

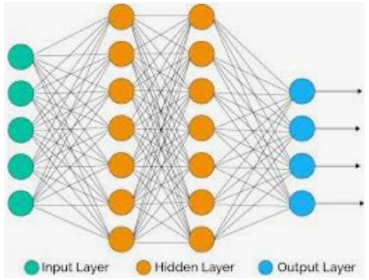
L. Trend
M. CenterNet
N. FCos

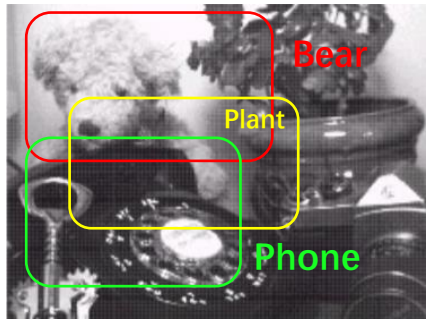
I. Two Stage Detection

Start from:









Input → Feature → Classic ML Tools → Target

Get Manually → Choose Carefully → Use Hopelessly

End up with:

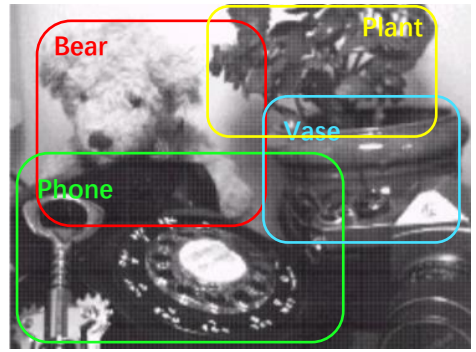
Input

Put In

CNN

Only Focus
On
This Part

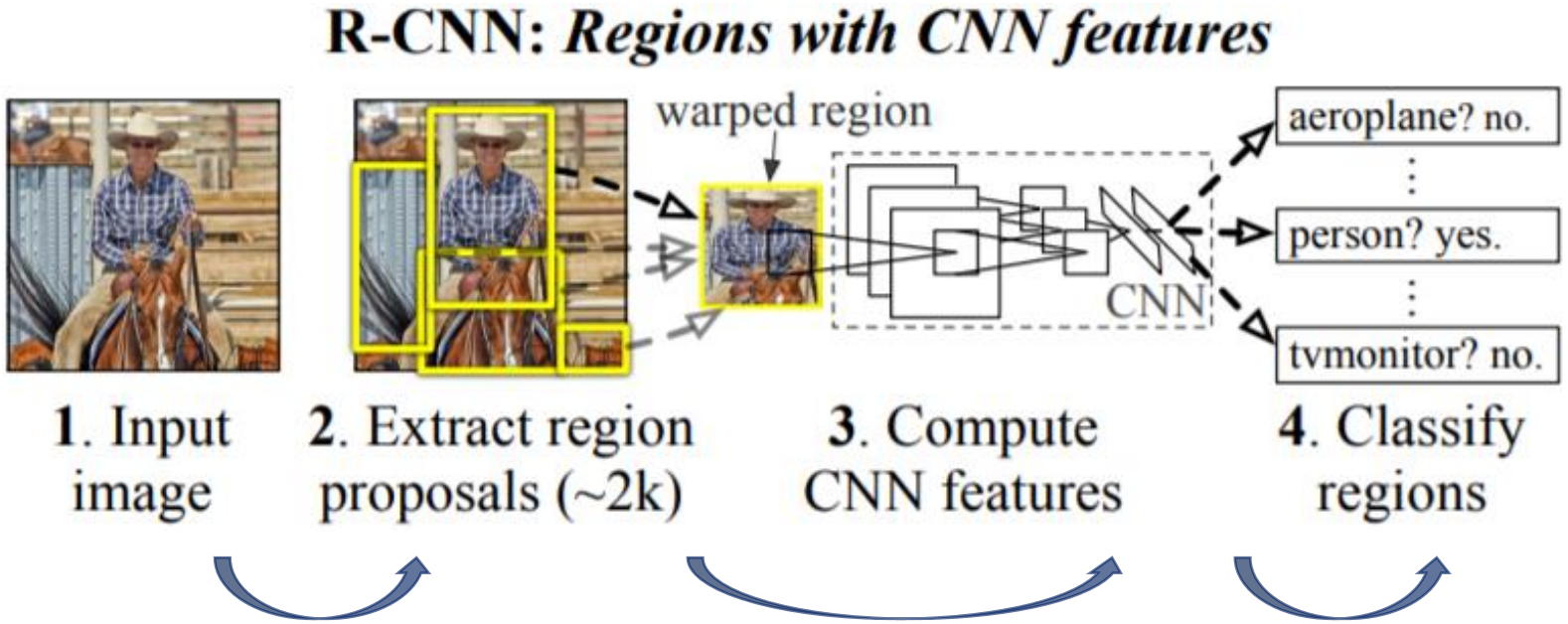
Use It



Target

I. Two Stage Detection

A. First Trial - RCNN [2014 Ross]:



A1. Region Proposal

A2. Feature Extraction

A3. Classification + Detection

A.1 Region Proposal

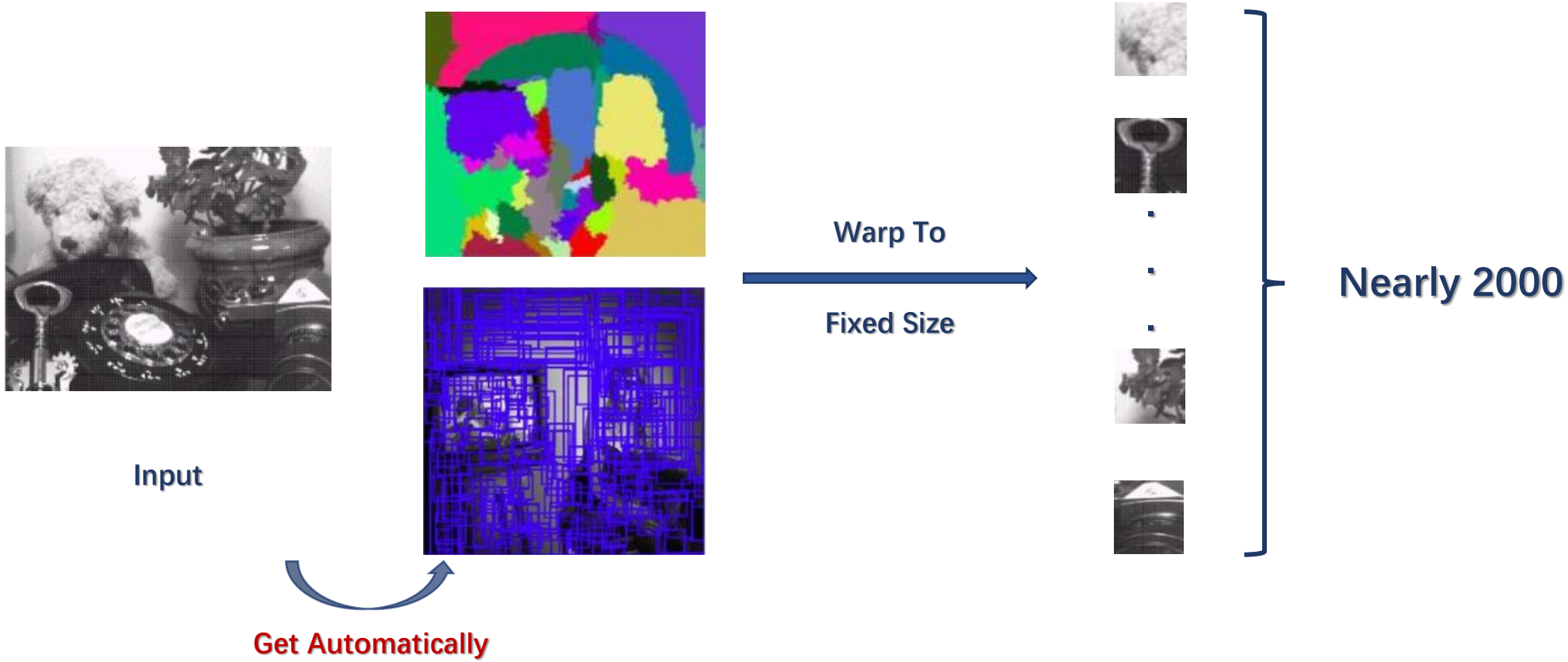
A.2 Feature Extraction

A.3 Cls + Det

I. Two Stage Detection

A. First Trial - RCNN [2014 Ross]:

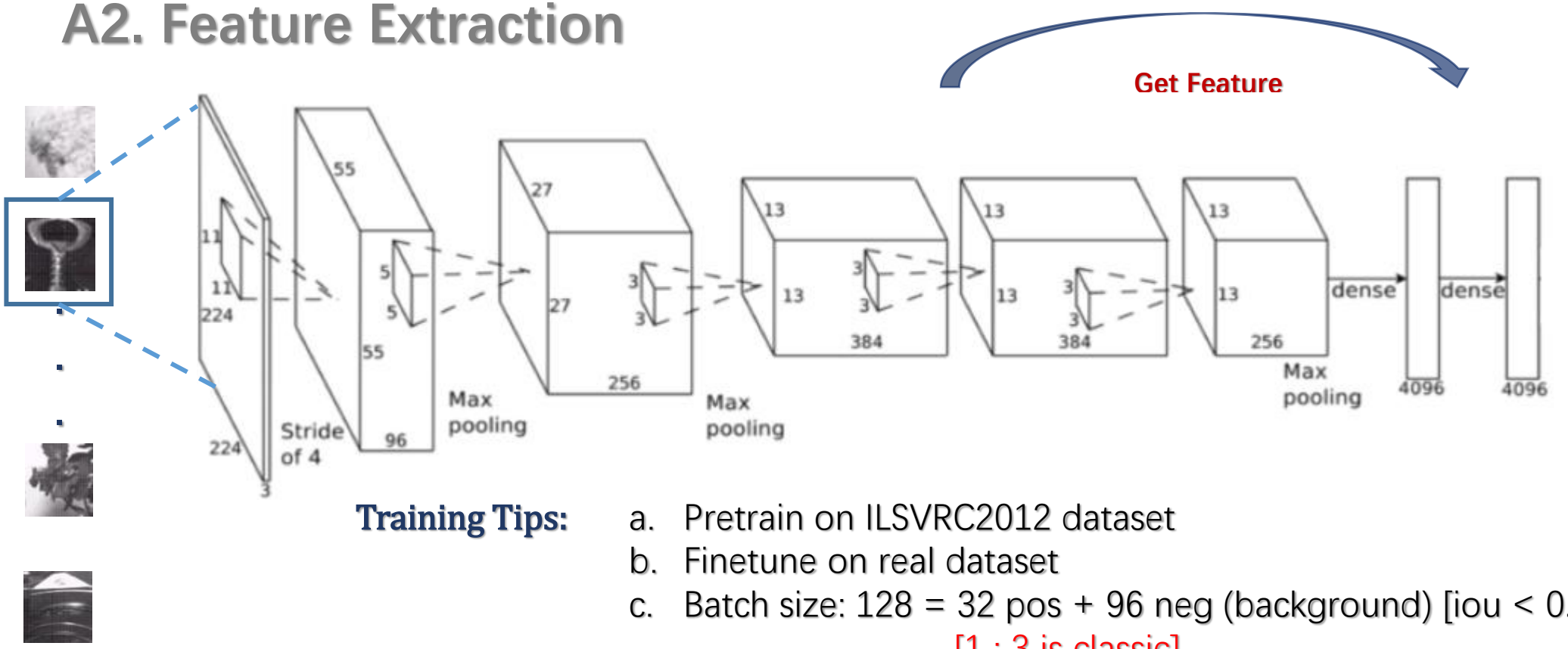
A1. Region Proposal: Selective Search



I. Two Stage Detection

A. First Trial - RCNN [2014 Ross]:

A2. Feature Extraction



Training Tips:

a. Pretrain on ILSVRC2012 dataset
b. Finetune on real dataset
c. Batch size: 128 = 32 pos + 96 neg (background) [iou < 0.5]
[1 : 3 is classic]

I. Two Stage Detection

A. First Trial - RCNN [2014 Ross]:

A3. SVM Classification + NMS + BBox Regression

a. SVM Classification

- Do classification for each class
- Pos: Ground truth
  Neg: iou < 0.3 [others ignored]

[Different from CNN part]

Q: Why iou threshold is different here comparing to training AlexNet?
A: We need more data when training CNN. But here more data may be harmful.

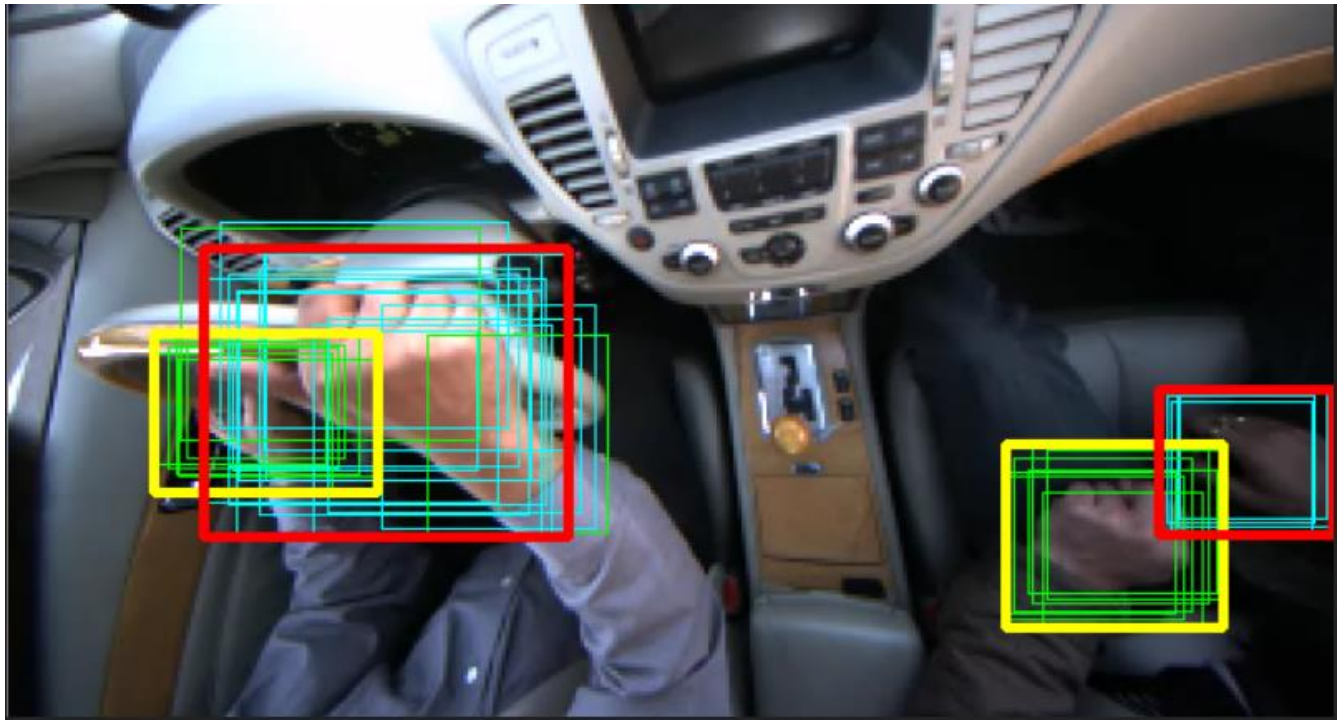
Q: Why use SVM? AlexNet has softmax at last, why not use directly?
A: AlexNet focuses on classification. Not accurate on localization
Use hard negative mining when using SVM

I. Two Stage Detection

A. First Trial - RCNN [2014 Ross]:

A3. SVM Classification + NMS + BBox Regression

b. NMS: Non-Maximum Suppression

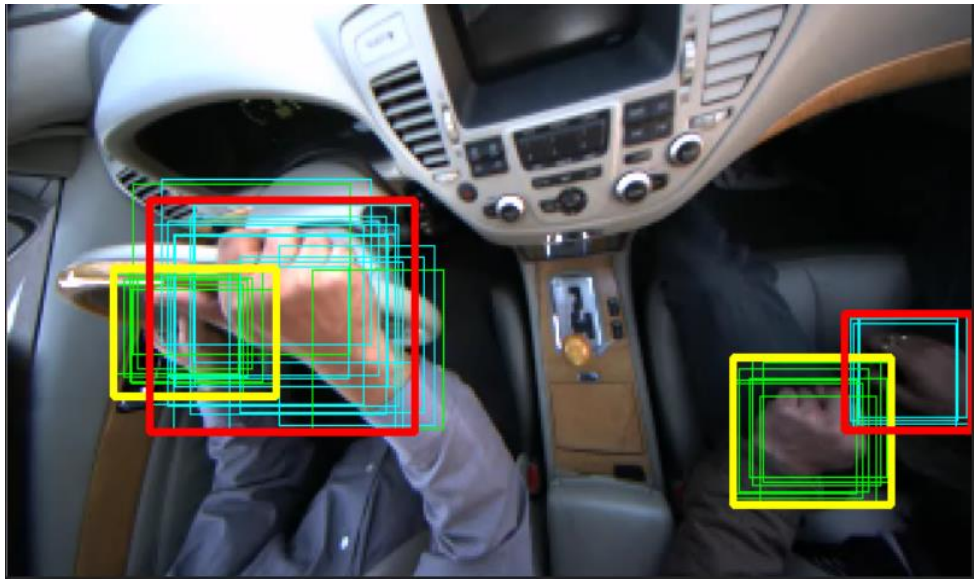


I. Two Stage Detection

A. First Trial - RCNN [2014 Ross]:

A3. SVM Classification + NMS + BBox Regression

b. NMS: Non-Maximum Suppression



- Sort bbox according to score [here class score]
- Get the 1st one. Remove those which has higher iou with it
- Then repeat the last step until there's no bbox

I. Two Stage Detection

A. First Trial - RCNN [2014 Ross]:

A3. SVM Classification + NMS + BBox Regression

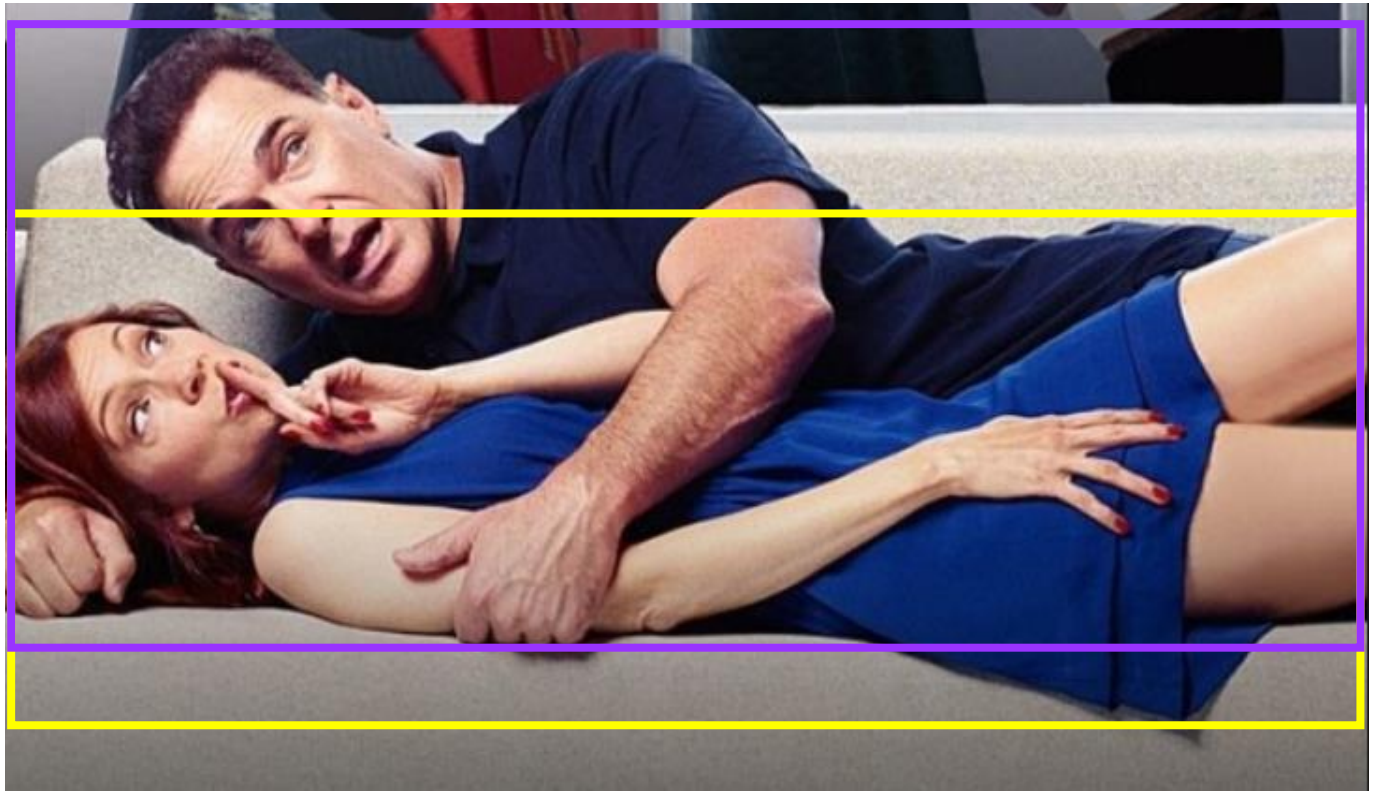
c. Bbox Regression

- Regression targets of (dx, dy, dw, dh)
- Coordinates need to be normalized
- We'll discuss in detail later

I. Two Stage Detection

A. First Trial - RCNN [2014 Ross]:

A*. Rethinking about NMS



I. Two Stage Detection

A. First Trial - RCNN [2014 Ross]:

A*. Rethinking about NMS

- If ▭ has higher score, we'll lose ▭
- If ▭ has higher score, we'll loss ▭
- That's awkward.

Let's Correct It!

I. Two Stage Detection

A. First Trial - RCNN [2014 Ross]:

A*. Rethinking about NMS

- **Brief history about NMS**
  - Traditional NMS [Here we use. Need to know how to code in C++/python]
  - Soft-NMS [2017, better to know]
  - IoU-Net [2018, Localization Confidence + PrROI Pooling, better to know]
  - Softer-NMS [2019, KL-Loss + Softer-NMS, better to know]

Important / Will cover / Remind

I. Two Stage Detection

A. First Trial - RCNN [2014 Ross]:

A*. Rethinking about NMS

- **Soft-NMS**

Input : $\mathcal{B} = \{b_1, .., b_N\}, \mathcal{S} = \{s_1, .., s_N\}, N_t$
$\mathcal{B}$ is the list of initial detection boxes
$\mathcal{S}$ contains corresponding detection scores
N_t is the NMS threshold

begin
  $\mathcal{D} \leftarrow \{\}$
  while $\mathcal{B} \neq$ *empty* **do**
    $m \leftarrow$ argmax $\mathcal{S}$
    $\mathcal{M} \leftarrow b_m$
    $\mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{M}; \mathcal{B} \leftarrow \mathcal{B} - \mathcal{M}$
    for b_i *in* $\mathcal{B}$ **do**
      if $iou(\mathcal{M}, b_i) \geq N_t$ **then**
        $\mathcal{B} \leftarrow \mathcal{B} - b_i; \mathcal{S} \leftarrow \mathcal{S} - s_i$
      end — NMS
      $s_i \leftarrow s_i f(iou(\mathcal{M}, b_i))$ — Soft-NMS
    end
  end
  return $\mathcal{D}, \mathcal{S}$
end

I. Two Stage Detection

A. First Trial - RCNN [2014 Ross]:

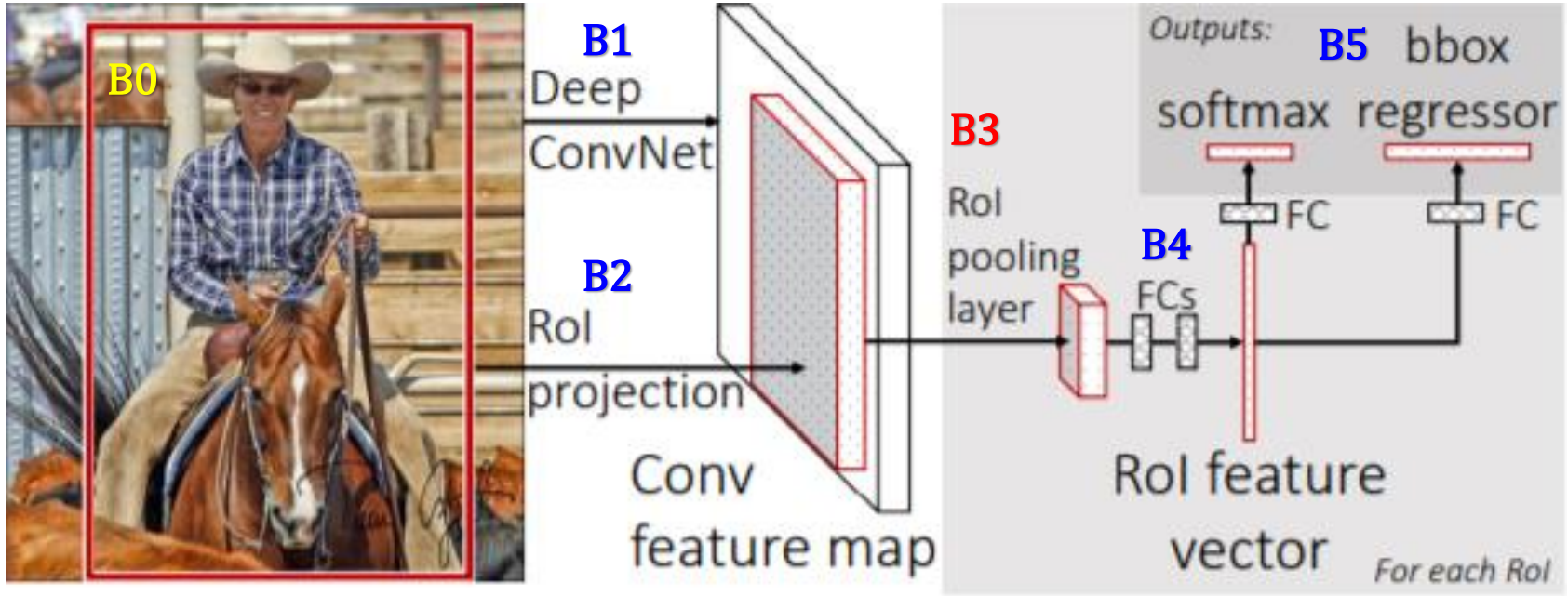
A4. RCNN Problems

- **Slow: need to run full forward pass of CNN for each region proposal**
- **SVMs and regressors are post-hoc: CNN features not updated in response to SVMs and regressors**
- **Complex multistage training pipeline**

Let's Fix It, Step by Step

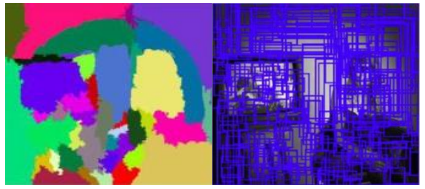
I. Two Stage Detection

B. Fast R-CNN [2015 Ross]:



I. Two Stage Detection

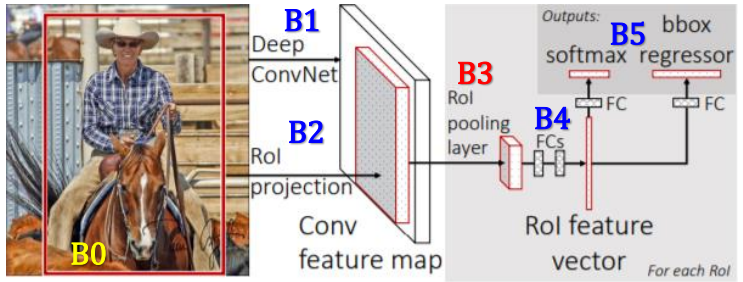
B. Fast R-CNN [2015 Ross]:



B0. Region Proposal: Same as RCNN

- 2k per image. **Record location for each ROI**

B1 & 2. Convolution & Projection



- Do Conv per ROI. Project location for each ROI
- 3 basic structures provided, use Vgg16 as an E.g.
- 4 max pooling, /16

B3. ROI Pooling

- Grid each ROI in feature map to fixed size, and do max pooling within each grid
- **So different size of feature maps can transfer into feature maps with same size.**

- bs = 2; 64 ROIs / image -> 128 proposals;
- ¼ pos & ¾ neg
- IoU > 0.5 pos, [0.1, 0.5) neg , [0. 0.1) hard neg
- Train on pos & neg. Test use neg. Retrain bad case

I. Two Stage Detection

B. Fast R-CNN [2015 Ross]:

B4. FC Layers

- FC layer cost a lot. Can use SVD to accelerate

B5. Multi-task Loss

- Classification: n + 1(background), softmax + cross entropy, L_{cls}
- Localization: Offset, SmoothL1, L_{loc}
- Total: $L(p, u, t^u, t^v) = L_{cls}(p, u) + \lambda[u \geq 1]\ L_{loc}(t^u, v)$

  p: predicted class score

  u: real class

  t^u: generated bbox [offset]

  t^v: real bbox

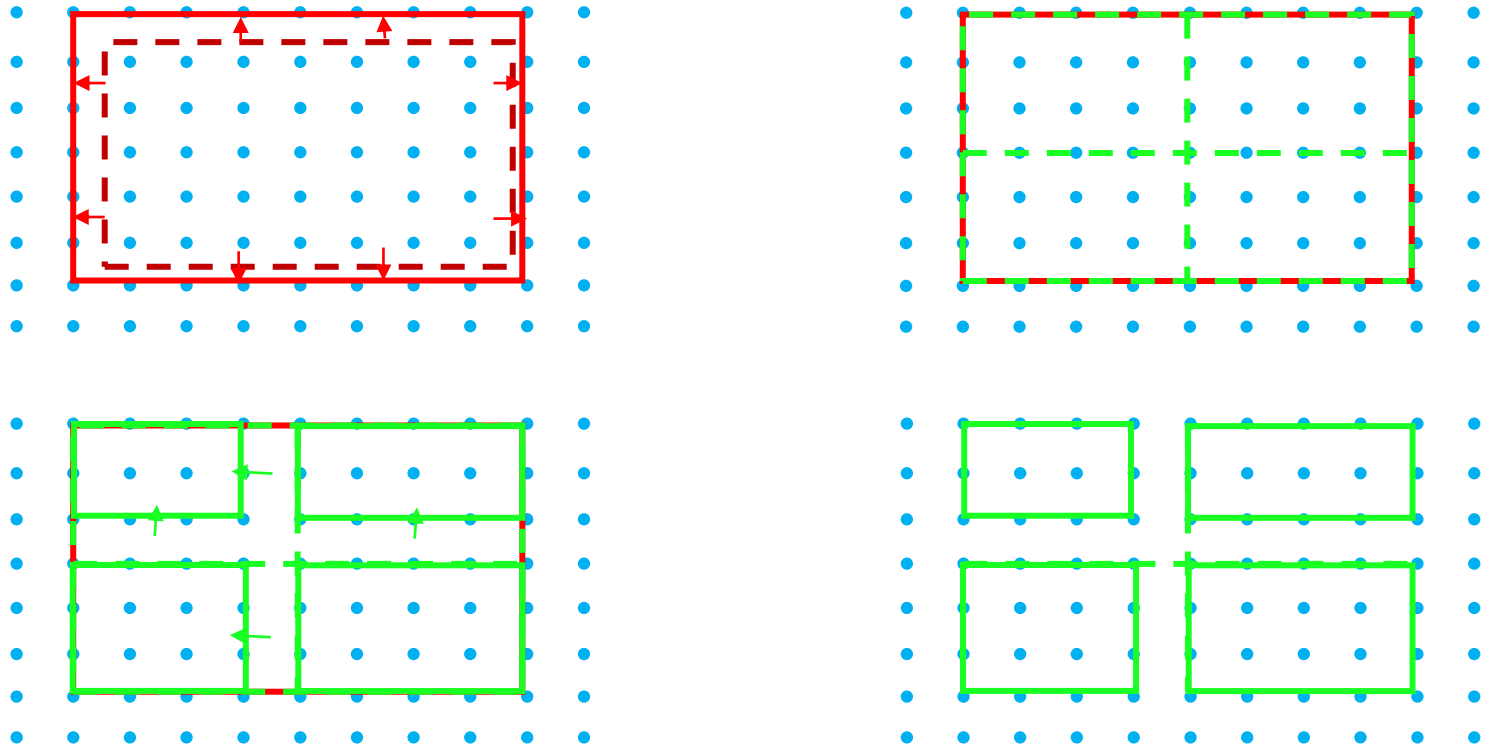
  $$\lambda = \begin{cases} 1, u \geq 1 \\ 0, u = 0 \end{cases}$$

SmoothL1 & Offset regression will be discussed in next topic

I. Two Stage Detection

B. Fast R-CNN [2015 Ross]:

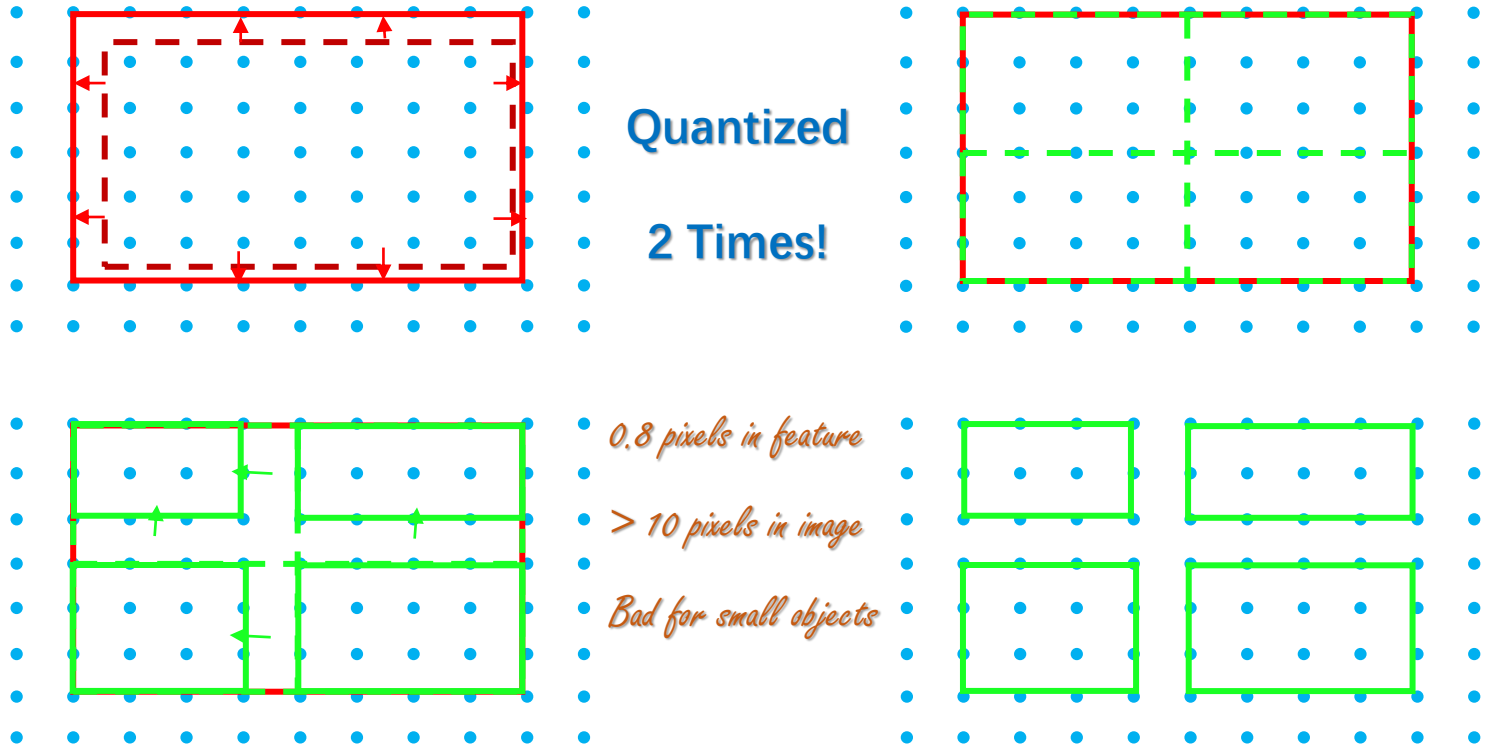
B*. Rethinking ROI Pooling



I. Two Stage Detection

B. Fast R-CNN [2015 Ross]:

B*. Rethinking ROI Pooling



I. Two Stage Detection

B. Fast R-CNN [2015 Ross]:

B*. Rethinking ROI Pooling

- **BP for ROI Pooling**

$$\frac{\partial L}{\partial x_i} = \sum_r \sum_j [i = i^*(r,j)] \frac{\partial L}{\partial y_{r,j}}$$

$y_{r,j}$: rth region, jth feature point

$i^*(r,j)$: the position where $y_{r,j}$ comes from

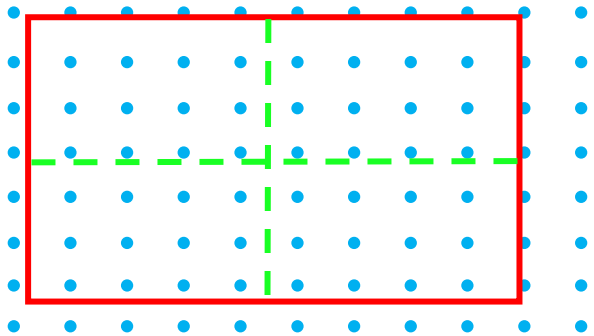
Just one point has loss passed by in each bin.

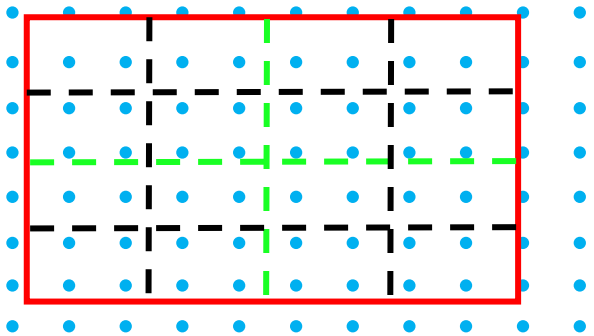
I. Two Stage Detection

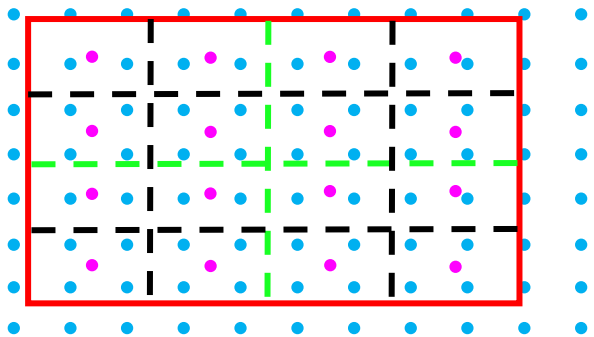
B. Fast R-CNN [2015 Ross]:

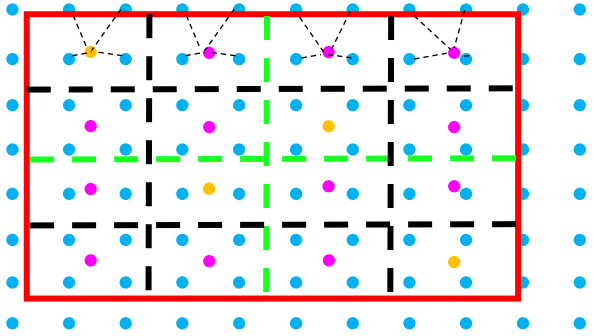
B*. Rethinking ROI Pooling

- ROI Align [2017, Kaiming]







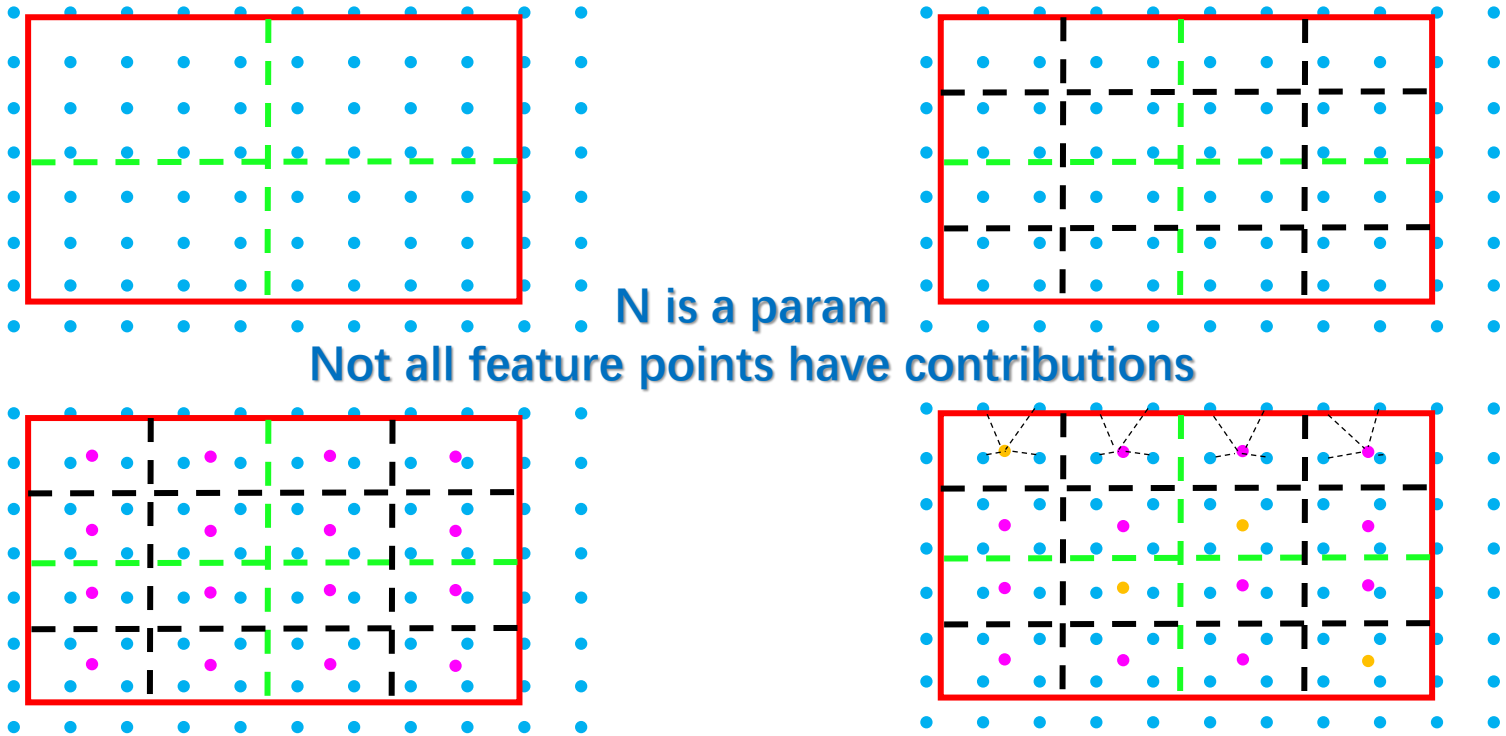


I. Two Stage Detection

B. Fast R-CNN [2015 Ross]:

B*. Rethinking ROI Pooling

- ROI Align [2017, Kaiming]



I. Two Stage Detection

B. Fast R-CNN [2015 Ross]:

B*. Rethinking ROI Pooling

- **ROI Align [2017, Kaiming] – BP**

$$\frac{\partial L}{\partial x_i} = \sum_r \sum_j [d(i, i^*(r,j))](1-\Delta h)(1-\Delta w)\frac{\partial L}{\partial y_{r,j}}$$

$d(x, y)$: distance between x & y

$\Delta w, \Delta h$: width & height difference of x_i & $x_{i^*(r,j)}$

$x_{i^*(r,j)}$: a float value interpolated by the forward pass

x_i: feature point before pooling

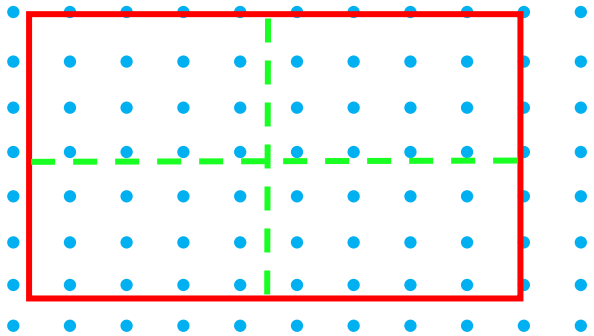
Only several points have losses passed by in each bin.

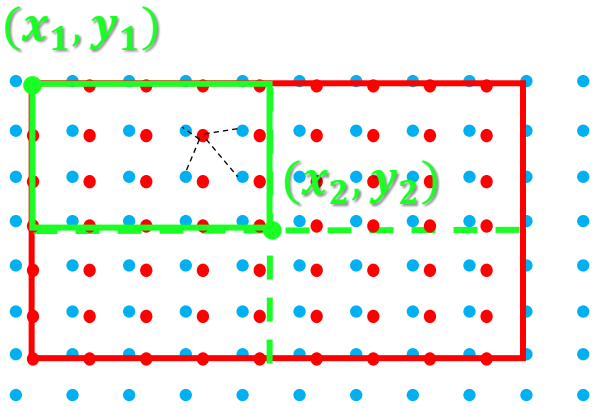
I. Two Stage Detection

B. Fast R-CNN [2015 Ross]:

B*. Rethinking ROI Pooling

- **Precise ROI Pooling [2018, IoU-Net]**





No Quantization

No Parameter

Pass losses for all

$$f(x,y) = \sum_{i,j} IC(x,y,i,j) * w_{i,j}$$

$$IC(x,y,i,j) = \max(0,1-|x-i|) + \max(0,1-|y-j|)$$

Do Pr ROI Pool for ▢:

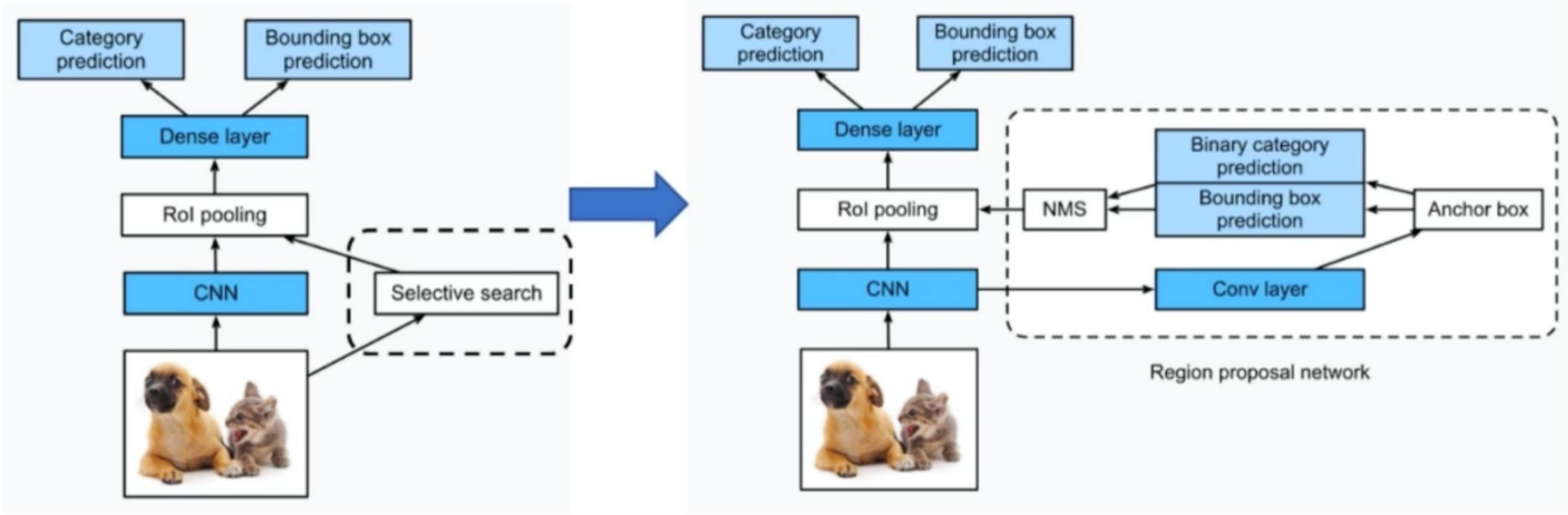
$$PrPool(bin,F) = \frac{\int_{y_1}^{y_2}\int_{x_1}^{x_2} f(x,y)dxdy}{(x_2-x_1)*(y_2-y_1)}$$

BP:

$$\frac{\partial PrPool(bin,F)}{\partial x_1} = \frac{PrPool(bin,F)}{x_2-x_1} - \frac{\int_{y_1}^{y_2} f(x,y)dy}{(x_2-x_1)*(y_2-y_1)}$$

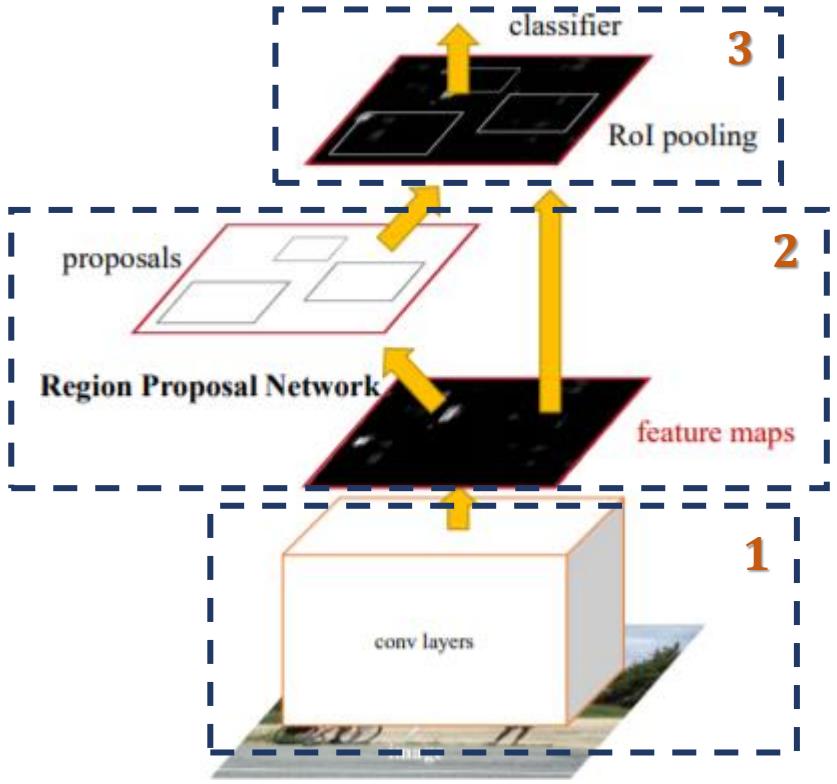
I. Two Stage Detection

Think:anything can be improved about fast R-CNN?



I. Two Stage Detection

C. Faster R-CNN [2015 Ren]:



C0. Structure

3. Fast RCNN: ROI + { Classification, Regression }

2. RPN

1. Backbone

Input

I. Two Stage Detection

C. Faster R-CNN [2015 Ren]:

C1. Backbone

- **Aim:** Extract feature integratedly
  - No need to extract feature per ROI
  - Use generated feature to generate proposals

- **Structure:** ZF / Resnet / VGG16 (13 conv + 13 ReLU + 4 Pooling)
  [Do have the sense of splicing network from now on]

- **Output:**

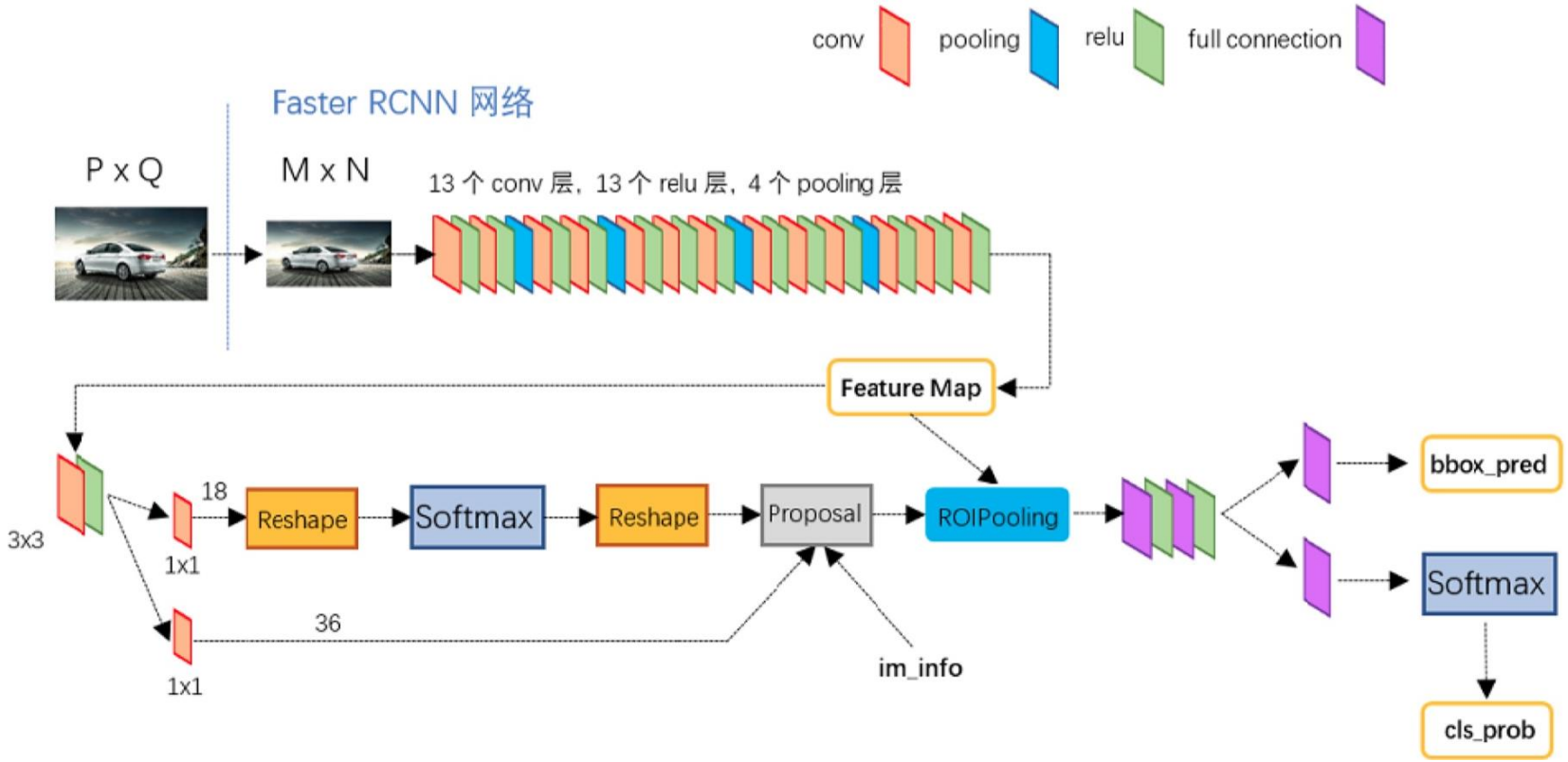
B	x	C	x	H/16	x	W/16
1	x	256	x	38	x	50

[Feature Map]

I. Two Stage Detection

C. Faster R-CNN [2015 Ren]:

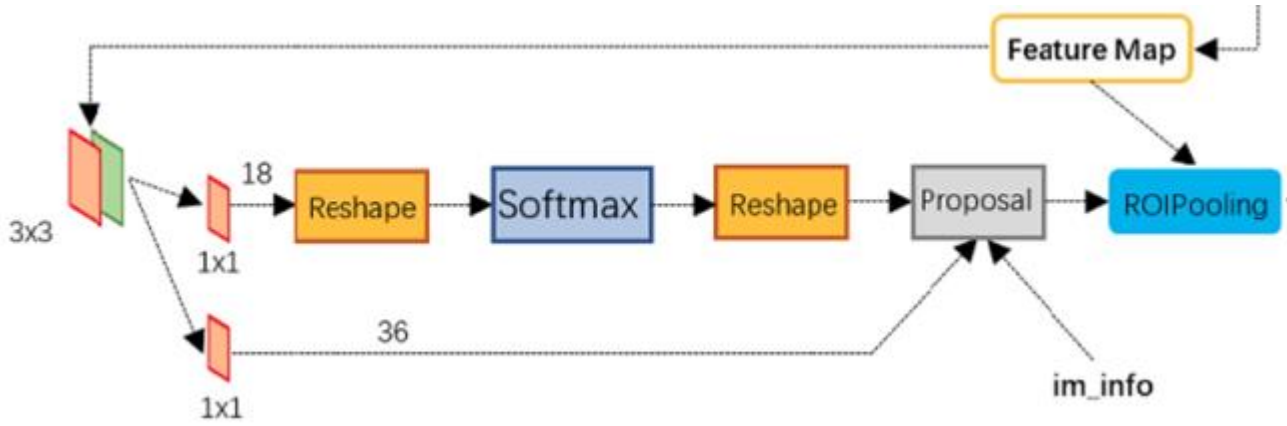
C1. Backbone



I. Two Stage Detection

C. Faster R-CNN [2015 Ren]:

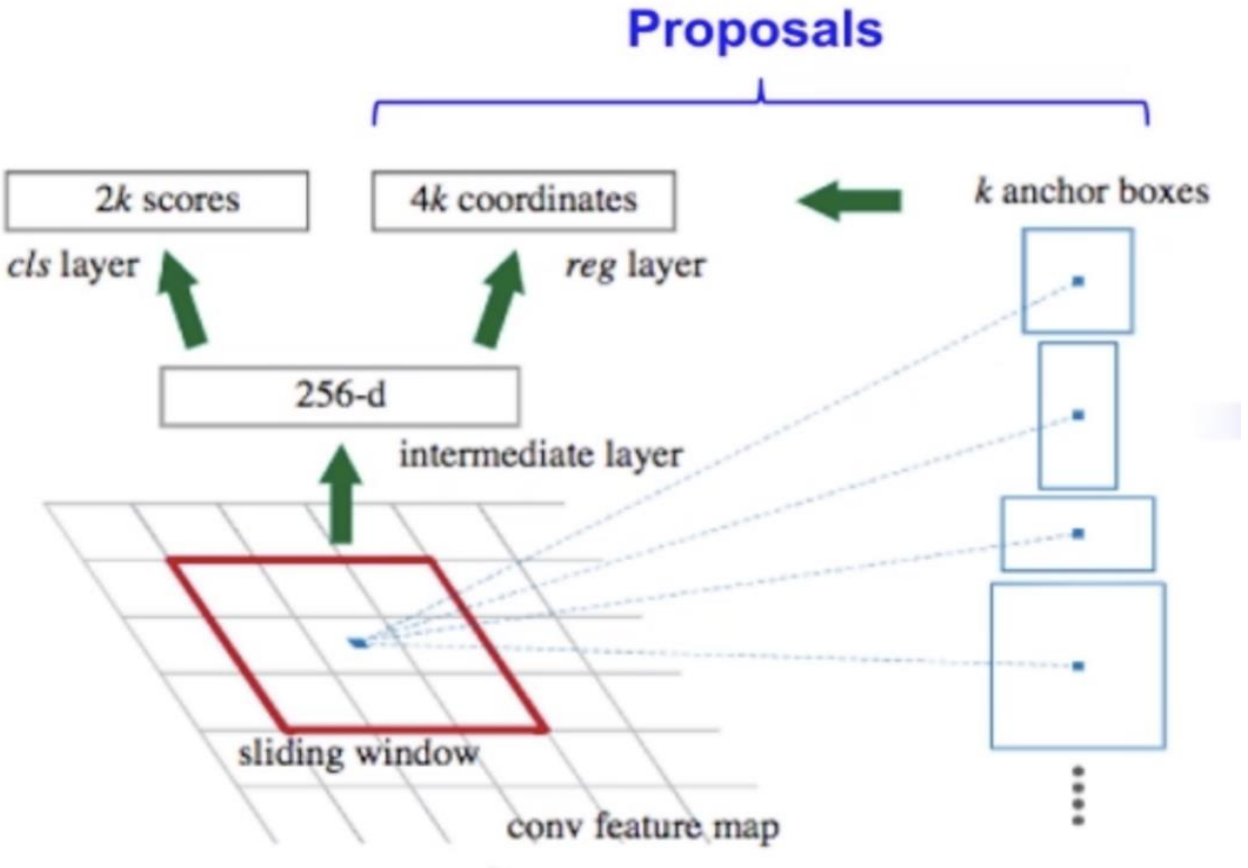
C2. RPN



I. Two Stage Detection

C. Faster R-CNN [2015 Ren]:

C2. RPN



I. Two Stage Detection

C. Faster R-CNN [2015 Ren]:

C2. RPN

- **Aim: Generate region proposal**
  [Real detected objects are coming from those RPs]

a. **It's the reason where the name "Two-Stage" coming from:**
   RPN + Bbox Regression

b. **It's the reason why some argue two-stage detection has better results then one-stage methods.**

I. Two Stage Detection

C. Faster R-CNN [2015 Ren]:

C2. RPN

- **Output of RPN:**
  a. **ROIs: 128 x 5**
     [0, x1, y1, x2, y2]->physical region proposal

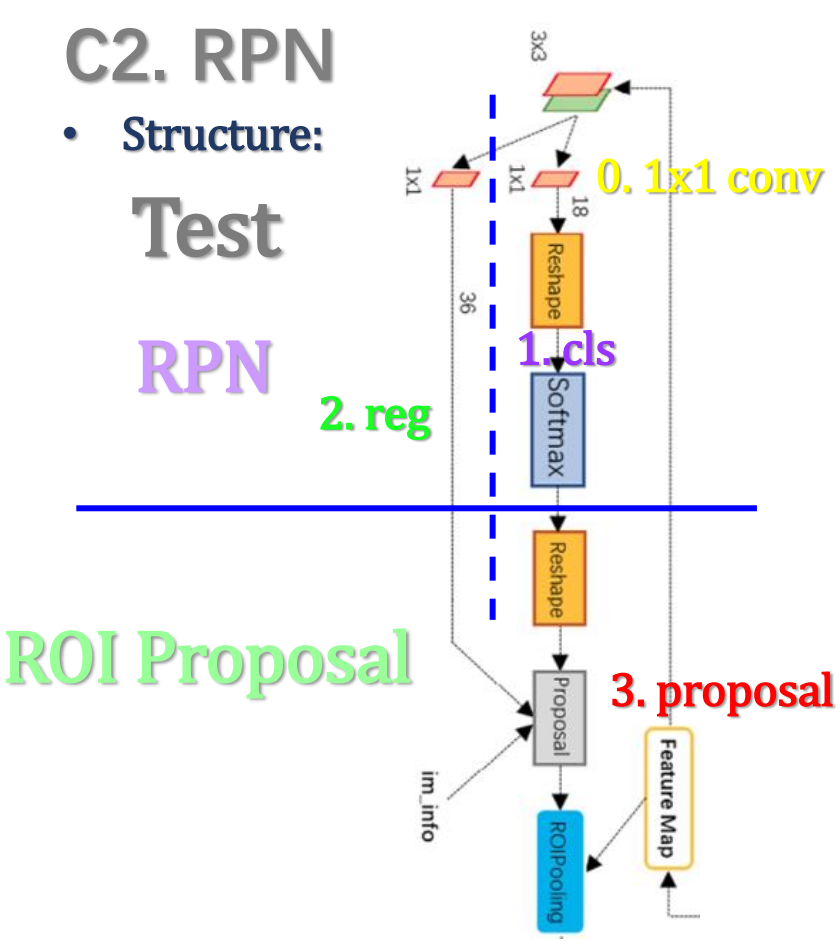
  b. **Label: 128**
     [0 ~ 20]->ROIs' classifications

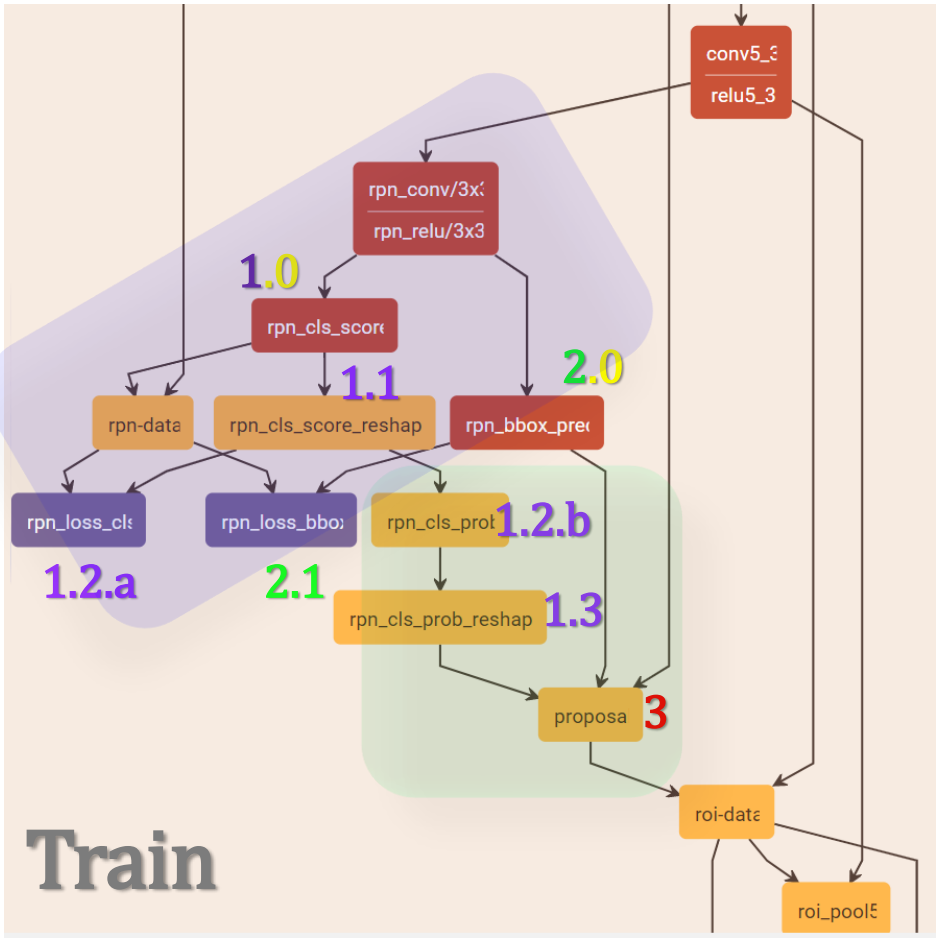
  c. **bbx_target: 128 x 84**
     [(20 + 1) x 4]->targets for bounding box regression

  d. **bbx_weight: 128 x 84**
     [0 0r 1]->weights for bbx_target when box regressing

I. Two Stage Detection

C. Faster R-CNN [2015 Ren]:





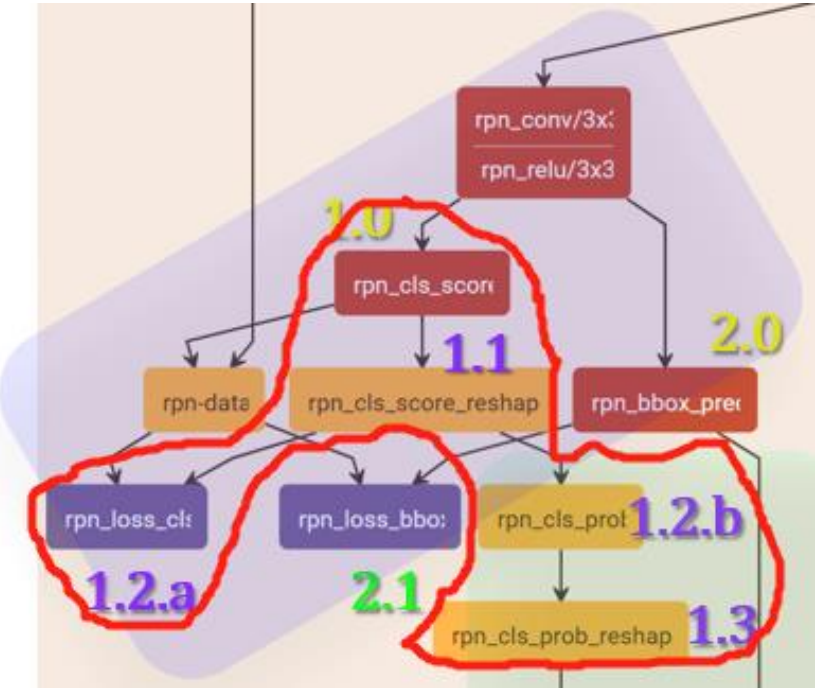
I. Two Stage Detection

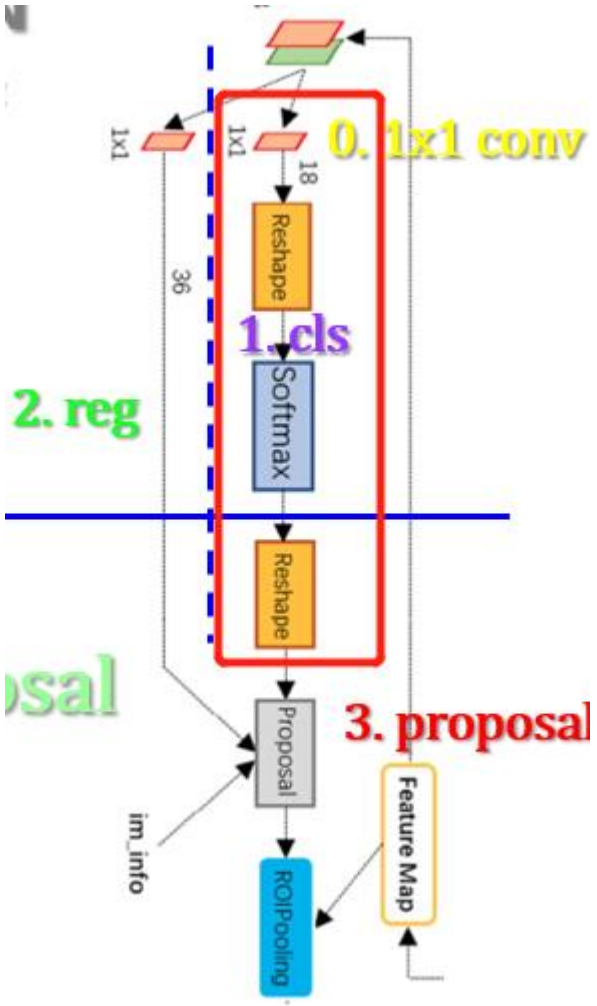
C. Faster R-CNN [2015 Ren]:

C2. RPN

- **Structure:**

C2.1: classification branch





I. Two Stage Detection

C. Faster R-CNN [2015 Ren]:

C2. RPN

- **Structure:**

C2.1.0: 1x1 conv

1 x 18 x 38 x 50

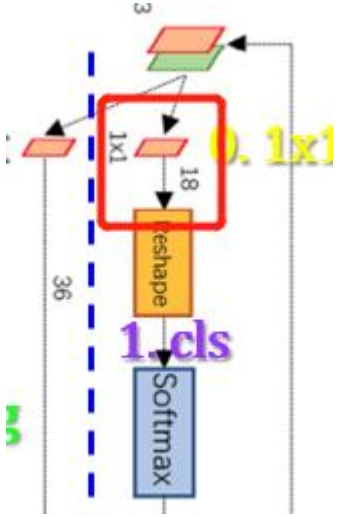
9 x 2

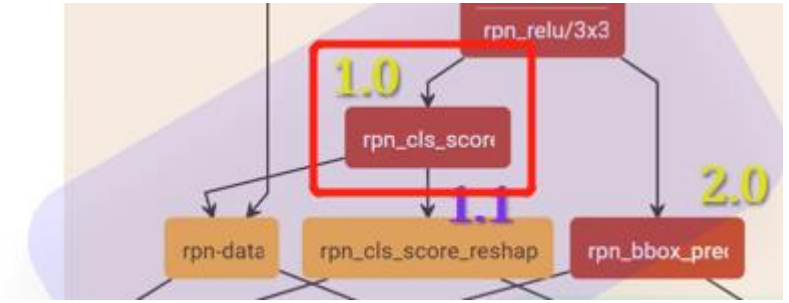
anchors 0|1

50

38

For each feature point,
We generate 9 anchors





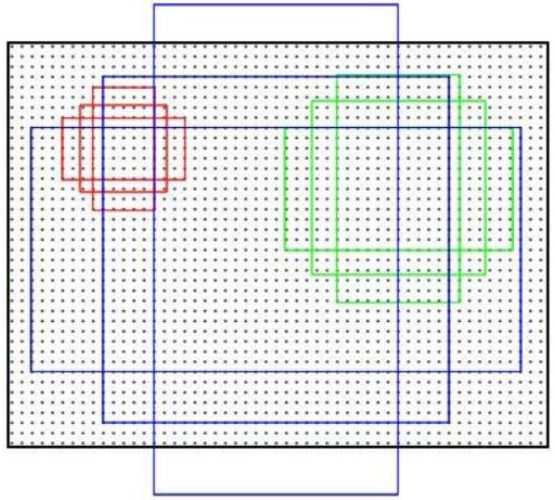
I. Two Stage Detection

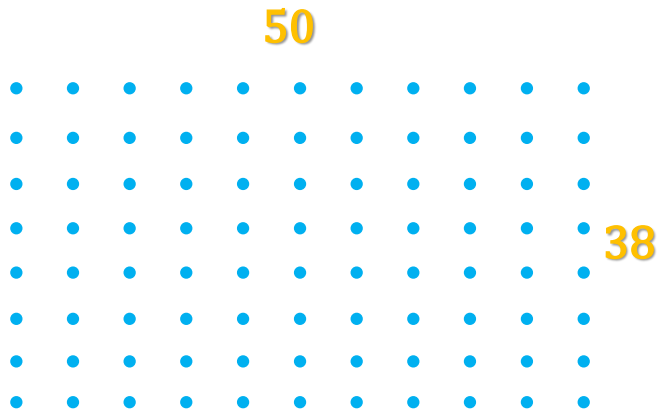
C. Faster R-CNN [2015 Ren]:

C2. RPN

- **Structure - Anchor:**

- ❑ **Represent an area in original image (a.k.a. where objects are)**
- ❑ **Have different scale & ratio to cover all kinds of objects**
- ❑ **9 in total: 3 scales x 3 ratios**
- ❑ **1 x 9 x 38 x 50 = 17100 anchors**
- ❑ **Coords of anchors are of original imgs**





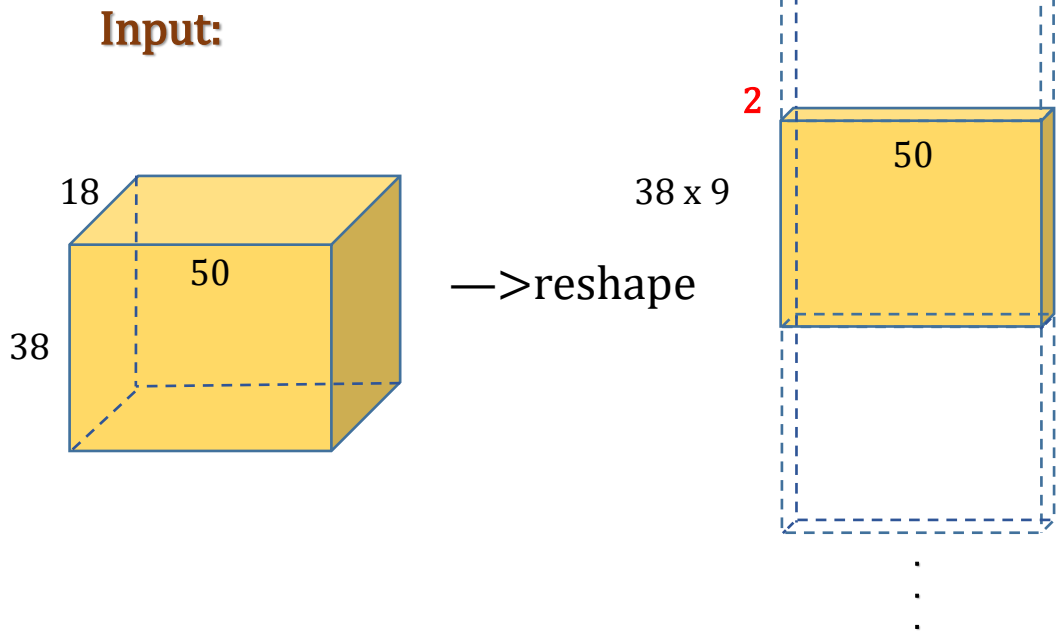
I. Two Stage Detection

C. Faster R-CNN [2015 Ren]:

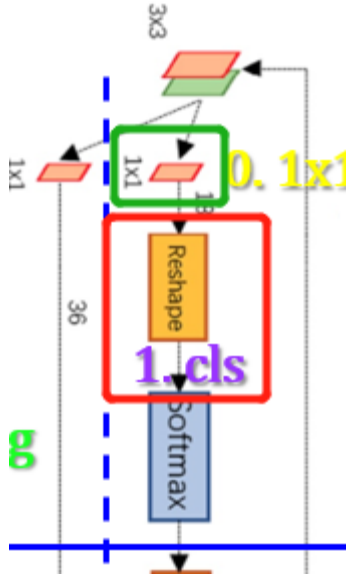
C2. RPN

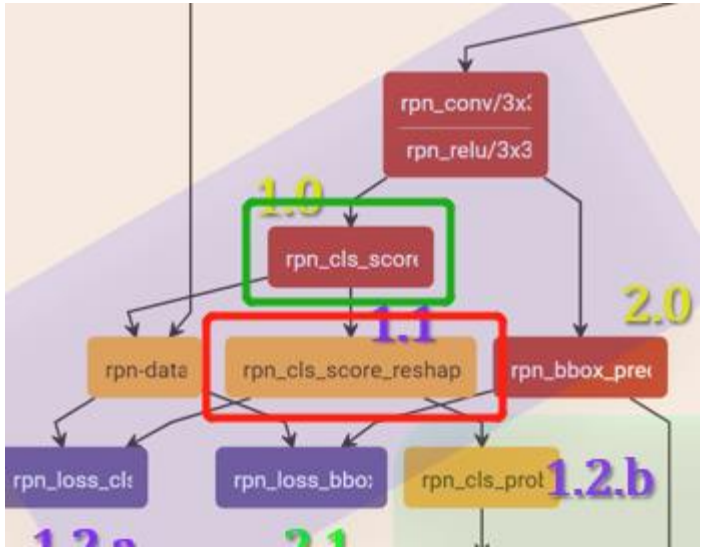
- **Structure:**

C2.1.1: cls reshape



This is for doing **Softmax** to get **fore/background**



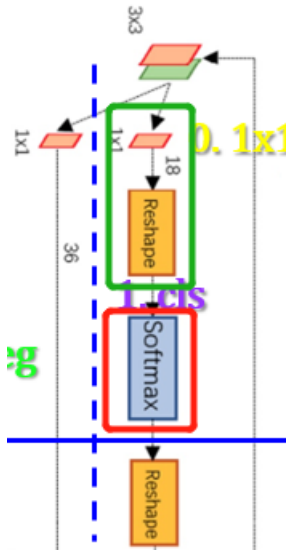


I. Two Stage Detection

C. Faster R-CNN [2015 Ren]:

C2. RPN

- **Structure:**



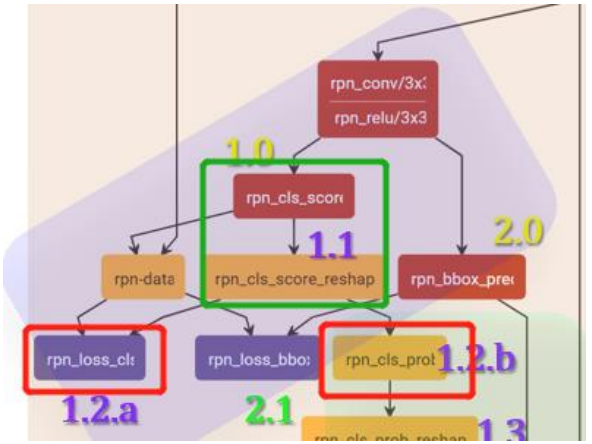
This is for doing Softmax
to get fore/background

C2.1.2: Softmax

2 branches: a. bp loss & b. no loss / get score

- a. 1 foreground & 0 background.
  Get loss with anchor and pass back.
  Use IoU

- b. Just do classification and get the score,
  Use the score to select proposal



I. Two Stage Detection

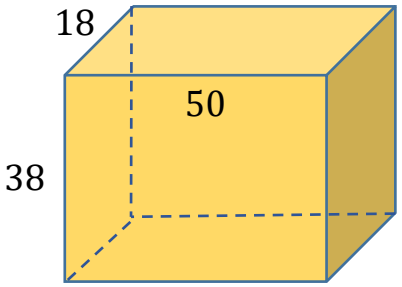
C. Faster R-CNN [2015 Ren]:

C2. RPN

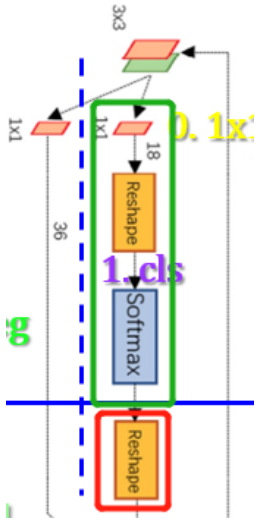
- **Structure:**

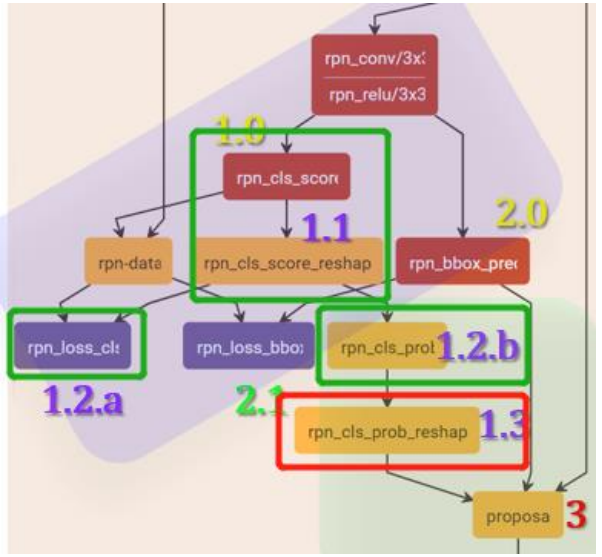
C2.1.3: cls reshape 2

Back to original shape



Each voxel represents the possibility of being a foreground or a background of an anchor





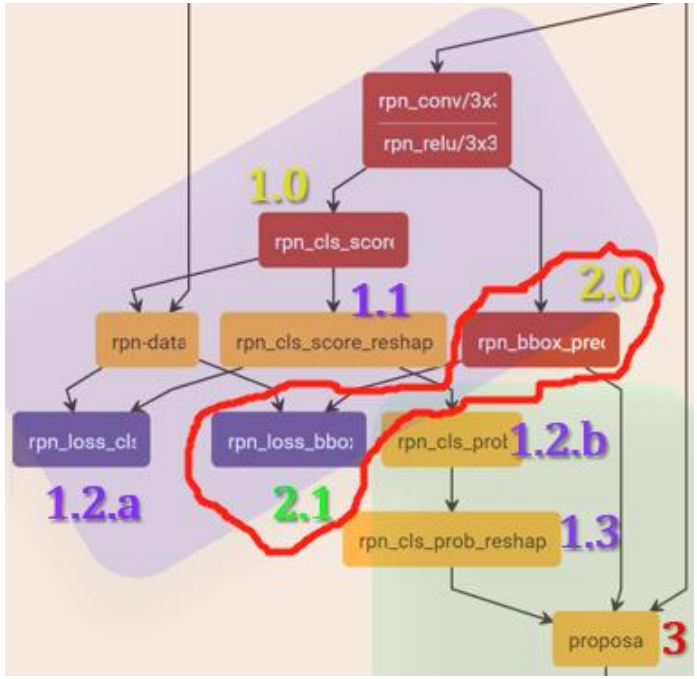
I. Two Stage Detection

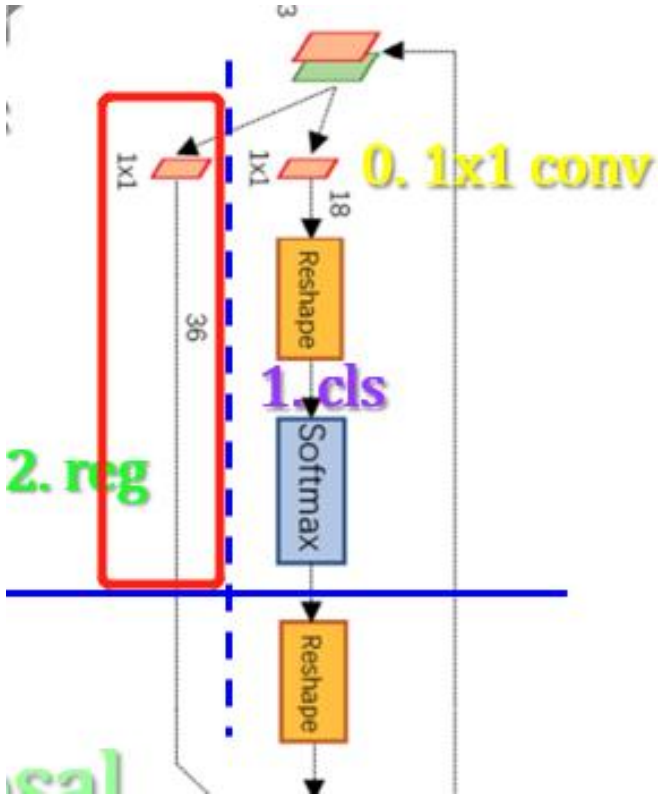
C. Faster R-CNN [2015 Ren]:

C2. RPN

- **Structure:**

C2.2: regression



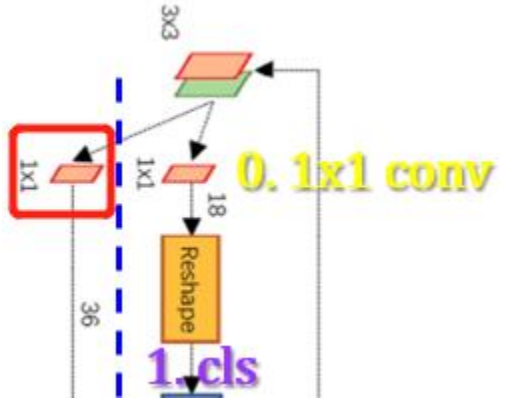


I. Two Stage Detection

C. Faster R-CNN [2015 Ren]:

C2. RPN

- **Structure:**



C2.2.0: 1x1 conv

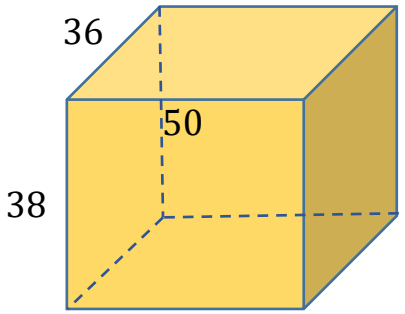
1 x 36 x 38 x 50

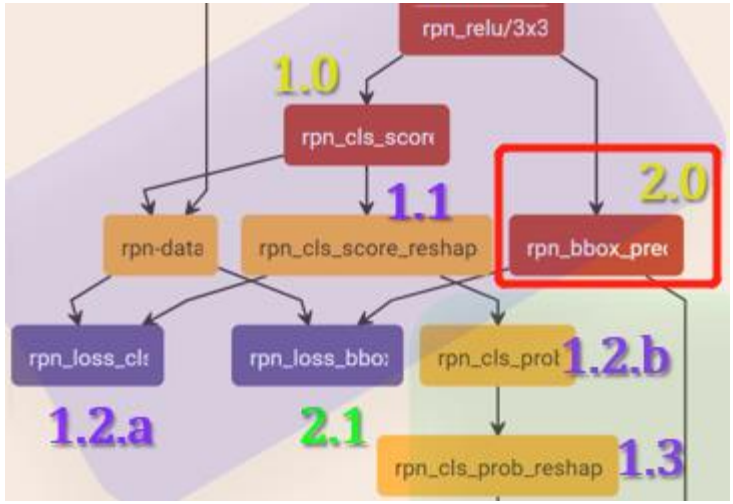
9 x 4

anchors 4 targets

Regress anchor to gt_bbox

Targets: proposals





I. Two Stage Detection

C. Faster R-CNN [2015 Ren]:

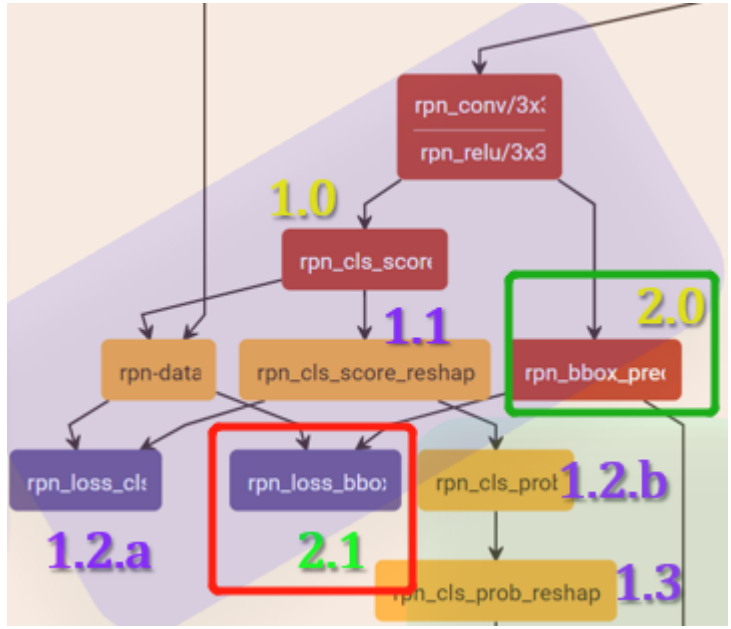
C2. RPN

- **Structure:**

C2.2.1: Smooth L1 Loss

- **Need to multiply the mask**
  - anchor = 1, mask = 1
  - anchor = 0, -1, mask = 0
- **Smooth L1 Loss**

$$f(x) = \begin{cases} \frac{(\sigma x)^2}{2}, & if\ x < 1/\sigma^2 \\ |x| - \frac{0.5}{\sigma^2}, & otherwise \end{cases}$$



I. Two Stage Detection

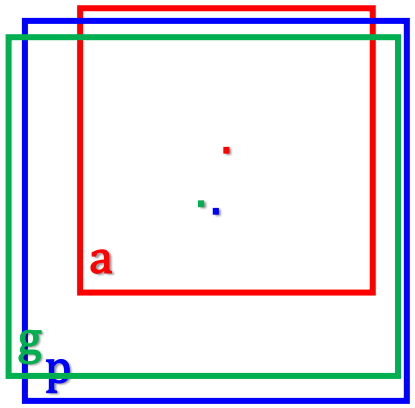
C. Faster R-CNN [2015 Ren]:

C2. RPN

- Structure:

C2.2.1: Smooth L1 Loss

➢ Regress (center) offset, NOT COORDINATES



We hope a -> g,
but actually a generates p.

So as long as p -> g,
We get a good result.

So as long as the offset of
p - a -> g – a
We get a good result

So here we hope $t_\cdot^p \rightarrow t_\cdot^g$

predicated bbox: x_p, y_p, w_p, h_p

anchor bbox: x_a, y_a, w_a, h_a

ground truth bbox: x_g, y_g, w_g, h_g

$$f(x) = \begin{cases} t_x^p = \frac{x_p - x_a}{w_a}, t_y^p = \frac{y_p - y_a}{h_a} \\ t_w^p = log\left(\frac{w_p}{w_a}\right), t_h^p = log\left(\frac{h_p}{h_a}\right) \\ t_x^g = \frac{x_g - x_a}{w_a}, t_y^g = \frac{y_g - y_a}{h_a} \\ t_w^g = log\left(\frac{w_g}{w_a}\right), t_h^g = log\left(\frac{h_g}{h_a}\right) \end{cases}$$

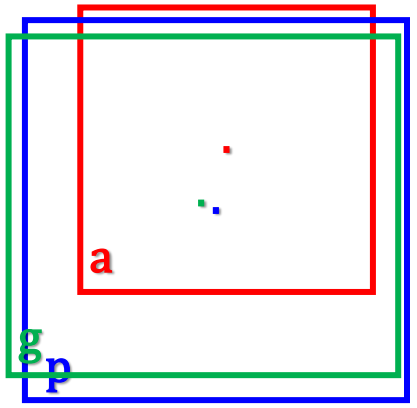
I. Two Stage Detection

C. Faster R-CNN [2015 Ren]:

C2. RPN

- **Structure:**

C2.2.1: Smooth L1 Loss



We hope **a** -> **g**,
but actually **a** generates **p**.

So as long as **p** ->**g**,
We get a good result.

So as long as the offset of
p - **a** -> **g** – **a**
We get a good result

So here we hope $t_\cdot^p \rightarrow t_\cdot^g$

$$f(x) = \begin{cases} t_x^p = \frac{x_p - x_a}{w_a}, t_y^p = \frac{y_p - y_a}{h_a} \\ t_w^p = log\left(\frac{w_p}{w_a}\right), t_h^p = log\left(\frac{h_p}{h_a}\right) \\ t_x^g = \frac{x_g - x_a}{w_a}, t_y^g = \frac{y_g - y_a}{h_a} \\ t_w^g = log\left(\frac{w_g}{w_a}\right), t_h^g = log\left(\frac{h_g}{h_a}\right) \end{cases}$$

Target: ◯ -> ◯ [regress offset]

predicated bbox: x_p, y_p, w_p, h_p ◯

anchor bbox: x_a, y_a, w_a, h_a ◯

ground truth bbox: x_g, y_g, w_g, h_g ◯

◯ : generated directly **FROM NET**

◯ : calculated [Encode]

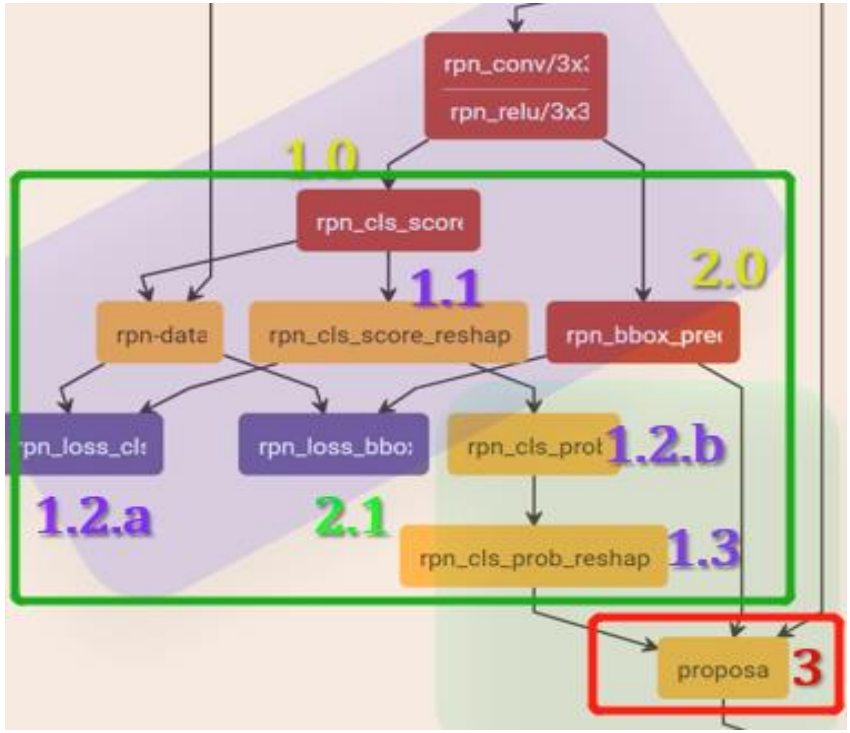
◯ : calculated – final outputs [Decode]

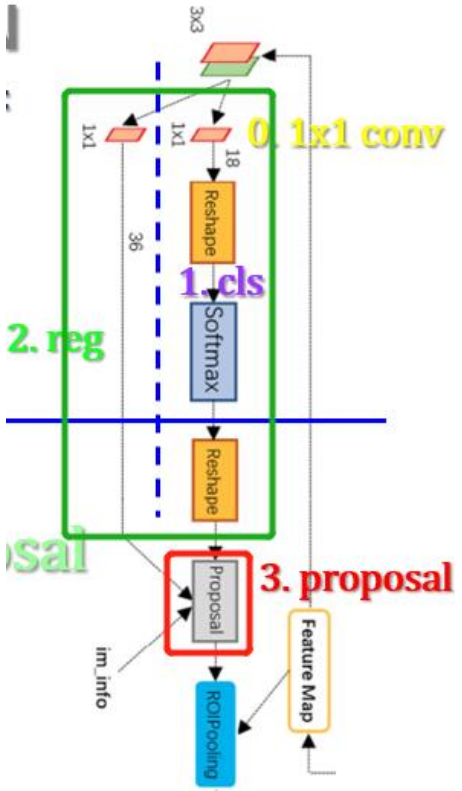
I. Two Stage Detection

C. Faster R-CNN [2015 Ren]:

C2. RPN

- **Structure:** C2.3: Proposal: Get proposal





I. Two Stage Detection

C. Faster R-CNN [2015 Ren]:

C2. RPN

- **Structure:**

C2.3: Proposal: Get proposal

- Regard anchor as FG when IoU > 0.7
- Regard anchor as FG when IoU < 0.3
- Regardless other anchors
- Then just keep 128: 0.25 fg + 0.75 bg anchors

I. Two Stage Detection

C. Faster R-CNN [2015 Ren]:

C3. After RPN

- **Same as Fast RCNN**

C4. How to train

- **Classic: alternatively 4 steps**
  - Use ImageNet finetune our RPN
  - Use trained proposal from step1 to train a Fast RCNN
  - Use detected results to initialize RPN training where we freeze the backbone layers but to train pure RPN-related layers
  - Finetune Fast RCNN-related layers only.
- **Other method:**
  - We can also train Faster RCNN as a whole in just one step

I. Two Stage Detection

D. Some Resources

D1. Faster RCNN Code Explanation

- Best code explanation so far [6 articles / explanation per line]

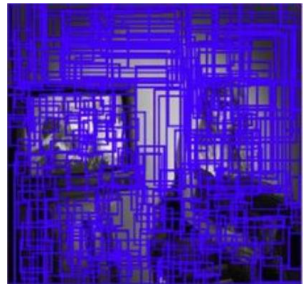
D2. Faster RCNN PyTorch

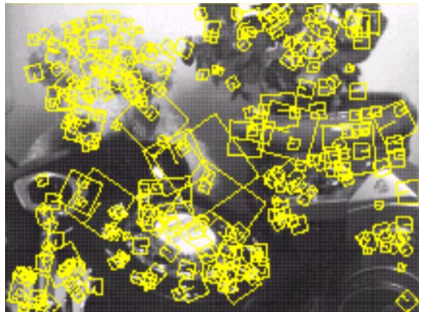
- Official Version [not for novice]
- Single Version [Chinese tutorial]

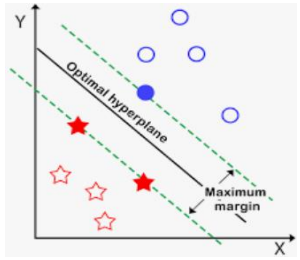
II. One Stage Detection

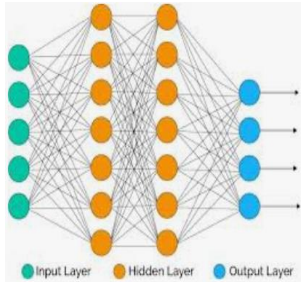
II. One Stage Detection

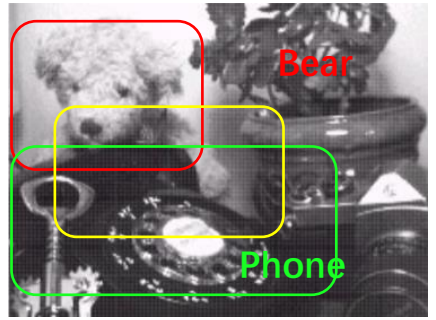
- Development of Detection Algorithm









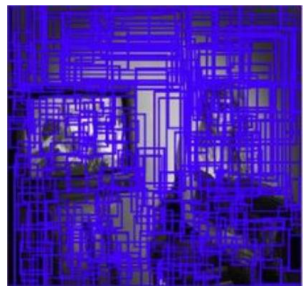


input → proposal → feature → historical ML → output

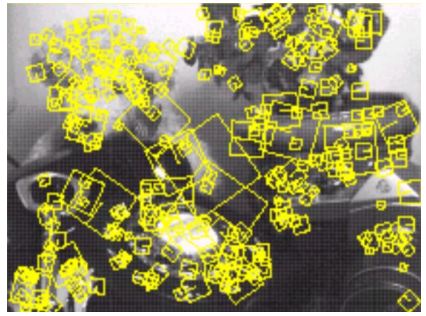
guess unsupervisedly | build manually | choose carefully | use hopelessly

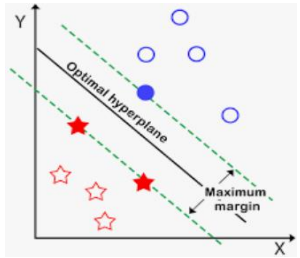
II. One Stage Detection

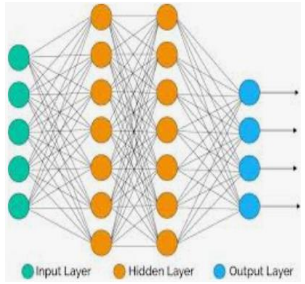
- Development of Detection Algorithm



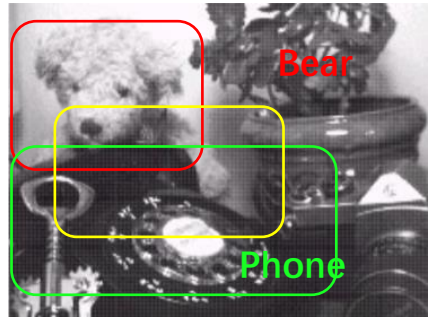
Stage I







Stage II

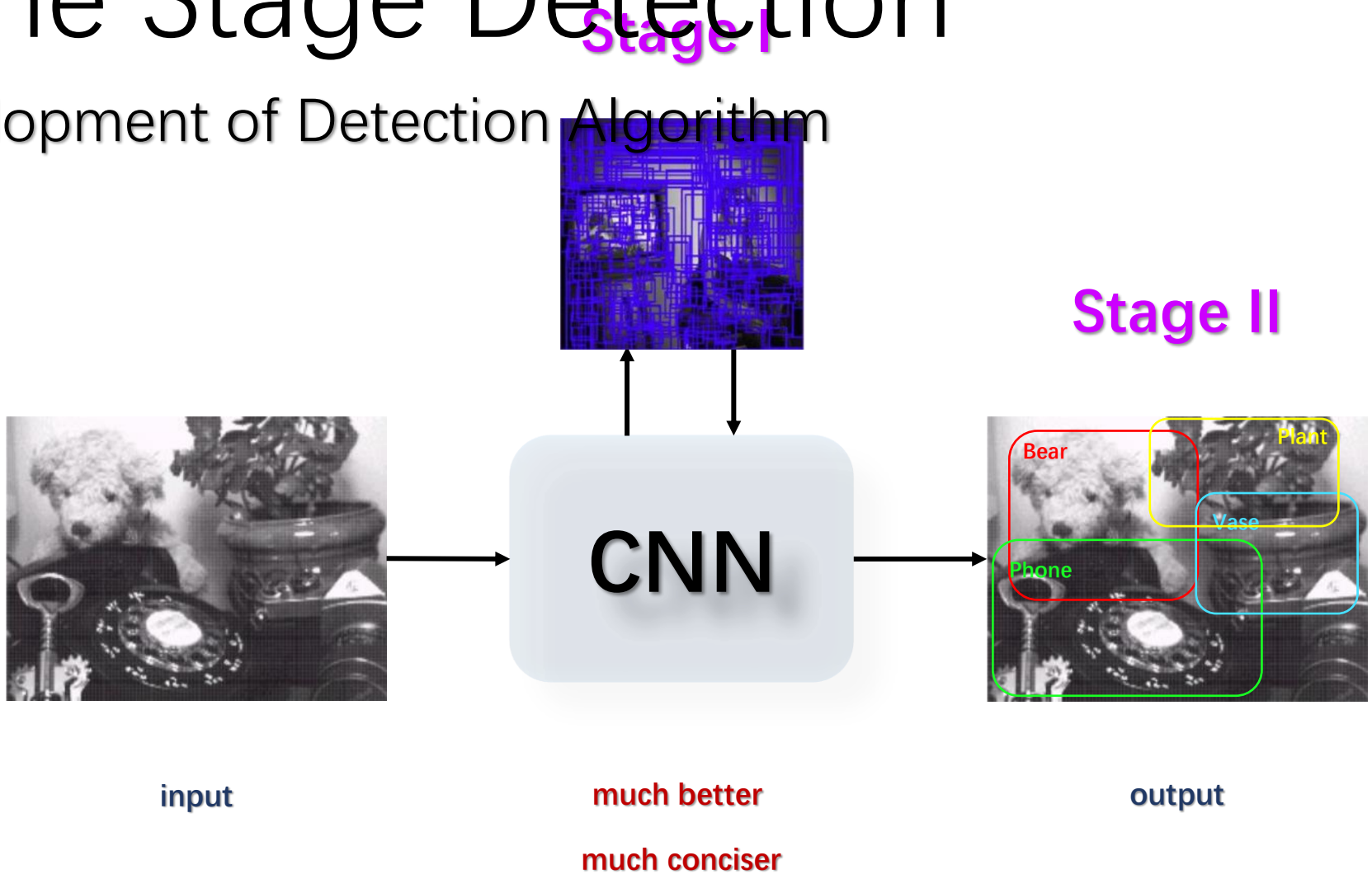


input → proposal → feature → historical ML → output

guess unsupervisedly → build manually → choose carefully → use hopelessly

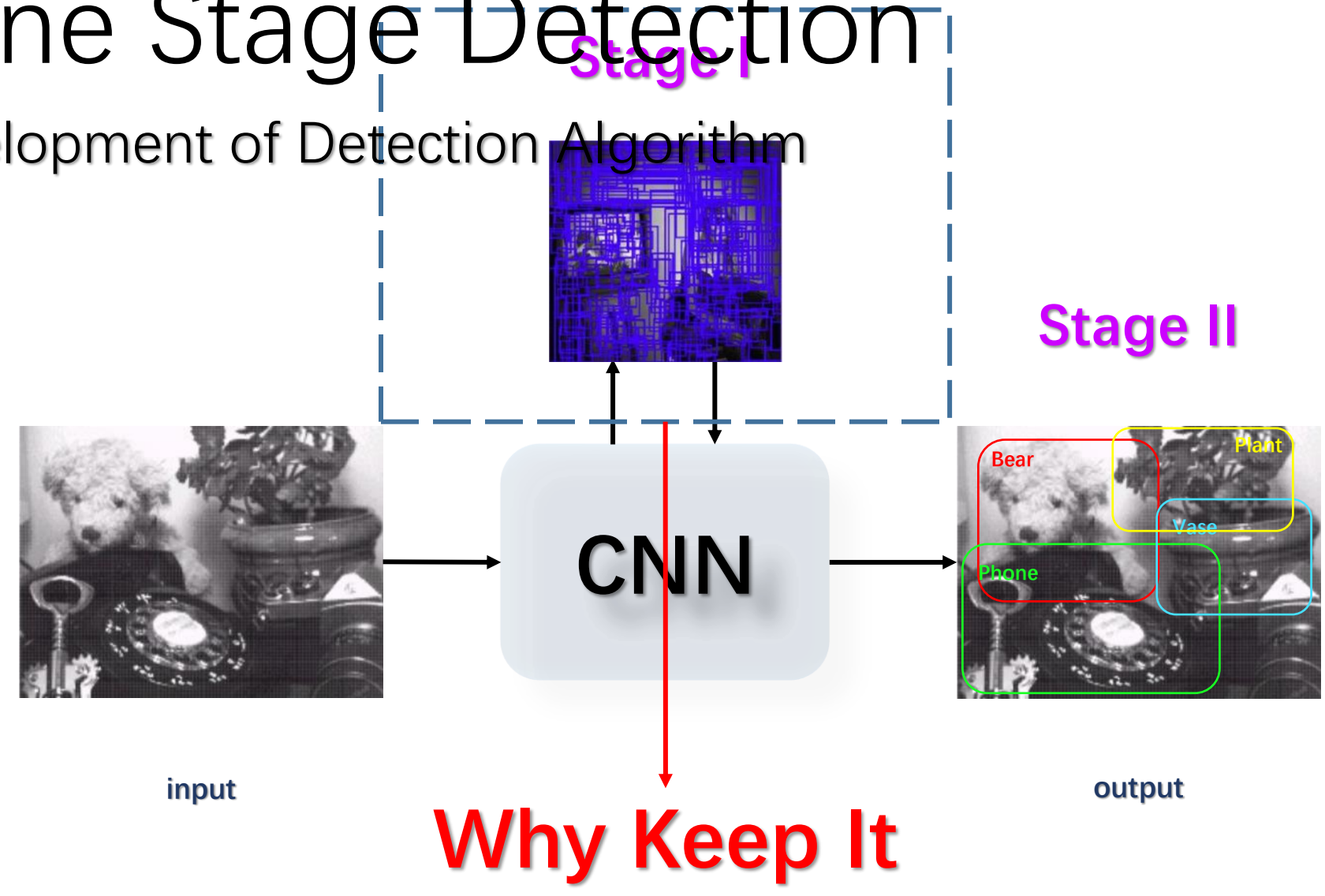
II. One Stage Detection

- Development of Detection Algorithm



II. One Stage Detection

- Development of Detection Algorithm



II. One Stage Detection

- Development of Detection Algorithm

Better?

II. One Stage Detection

- Key Point of One-Stage Detection

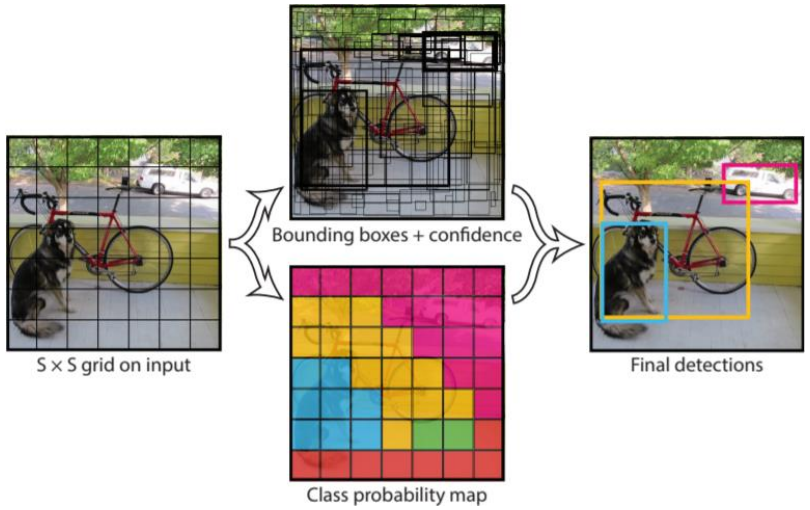
  - Inherit the end-to-end ideas from two-stage detection
  - Remove redundant region proposals

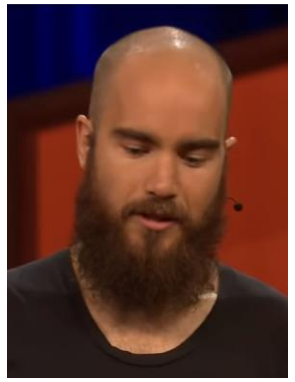
  - How to keep the accuracy?

II. One Stage Detection

E. Yolo V1 [2015, Joseph]

E0. You Only Look Once!





- Really fast (18 faster rcnn [ZF] vs 45 yolo)
- Becoming prevalent (62.1 mAP vs 63.4)

II. One Stage Detection

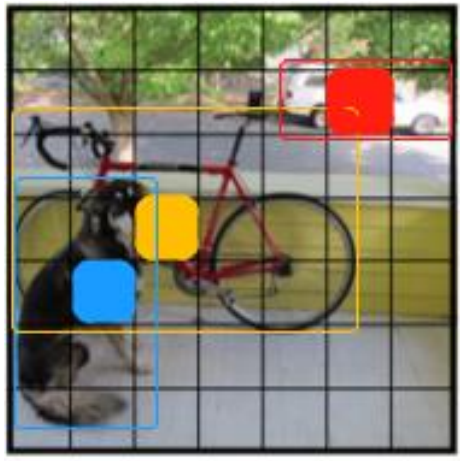
E. Yolo V1 [2015, Joseph]

E1. Procedure

- Grid an image into S x S cells [448 x 448 -> 7 x 7]

One cell will be responsible for predicting an object as long as an object's center locating in that cell.

S × S grid on input



S × S grid on input

II. One Stage Detection

E. Yolo V1 [2015, Joseph]

E1. Procedure

- Each cell predicts B bounding box with a confidence

Bounding Box: x, y, w, h (center)

Confidence: $P_r(object) \cdot IoU_{pred}^{truth}$

Final output tensor: S x S x (5 * B + C)

[7 x 7 x (5 * 2 + 20)] -> v1

II. One Stage Detection

E. Yolo V1 [2015, Joseph]

E2. Loss Function

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \quad (1)$$

2.1

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \quad (2)$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \quad (3)$$

2.2

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \quad (4)$$

2.3

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2 \quad (5)$$

$i: 0 \sim (S^2 - 1)$ [iterate each grid cell (0~48)]

$j: 0 \sim (B - 1)$ [iterate each bbox (0~2)]

$\mathbb{1}_{ij}^{\text{obj}}$ & $\mathbb{1}_{ij}^{\text{noobj}}$:

0	0	0	0	0	0	0
0	0	0	0	0	1	0
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

1	1	1	1	1	1	1
1	1	1	1	1	0	1
1	1	1	1	1	1	1
1	1	0	1	1	1	1
1	0	1	1	1	1	0
1	1	1	1	1	1	1
1	1	1	1	1	1	1

For $\mathbb{1}_{ij}^{\text{obj}}$, we have B predictions in each cell, only the one with largest IoU shall be labeled as 1.

II. One Stage Detection

E. Yolo V1 [2015, Joseph]

E2. Loss Function

E2.1: Coordinate loss

$$2.1 \left\{ \begin{aligned} &\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \quad (1) \\ &+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \quad (2) \end{aligned} \right.$$

x, y: predicated bbox center,
w, h: predicated bbox width & height

$\hat{x}, \hat{y}$: labeled bbox center,
$\hat{w}, \hat{h}$: labeled bbox width & height

$\sqrt{w}, \sqrt{h}$: Suppress the effect for larger bbox

λ_{coord}: 5, because there's only 8 dimensions. Too less comparing to other losses
Weighted loss essentially.

II. One Stage Detection

E. Yolo V1 [2015, Joseph]

E2. Loss Function

E2.2: Confidence loss

$\widehat{C}_i$: confidence score [IoU] of predicated and labeled bbox

C_i : predicated confidence score [IoU] generated from networks

2.2 (

$$+\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{\text{obj}}\left(C_i-\hat{C}_i\right)^2 \quad (3)$$

$$+\lambda_{\text{noobj}}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{\text{noobj}}\left(C_i-\hat{C}_i\right)^2 \quad (4)$$

Note: $\widehat{C}_i$ in (4) is 0

λ_{noobj}=0.5, because there're so many non-object bboxes

Train: $confidence=P_r(object)\cdot IoU_{pred}^{truth}$

$\mathbb{1}_{ij}^{\text{obj}}$ $\widehat{C}_i$

Test: Individual box confidence prediction:

$confidence=P_r(cls_i/obj)P_r(obj)\cdot IoU_{pred}^{truth}=P_r(cls_i)\cdot IoU_{pred}^{truth}$

II. One Stage Detection

E. Yolo V1 [2015, Joseph]

E2. Loss Function

E2.3: Classification Loss

Each sell will only predict 1 object, which is decided by the bbox with largest IoU

* Don't forget to do NMS after generating bboxes

$$2.3 \quad + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \quad (5)$$

II. One Stage Detection

E. Yolo V1 [2015, Joseph]

E3. Pros & Cons

E2.3: Classification Loss

Pros:

- One stage, really fast

Cons:

- Bad for crowed objects [1 cell 1 obj]
- Bad for small objects
- Bad for objects with new width-height ratio
- No BN

II. One Stage Detection

F. Yolo V2 [2016, Joseph]

F1. Improvements comparing to V1

F1.1: Add BN

F1.2: High Resolution Classifier [Focusing on backbone]

a. Train on ImageNet (224 x 224) // Model trained on small images may not be good

b. Resize & Finetune on ImageNet (448 x448) // So we finetune the model on larger images

c. Finetune on dataset // To let the model be used to larger images

d. We get 13 x 13 feature maps finally

F1.3: We Use Anchors [We'll talk it later]

II. One Stage Detection

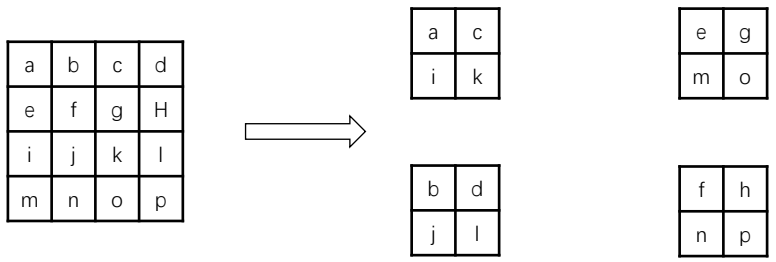
F. Yolo V2 [2016, Joseph]

F1. Improvements comparing to V1

F1.4: Fine-Grained Features

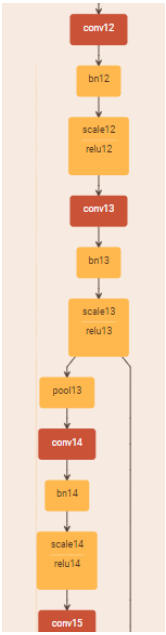
a. Lower features are concatenated directly to higher features

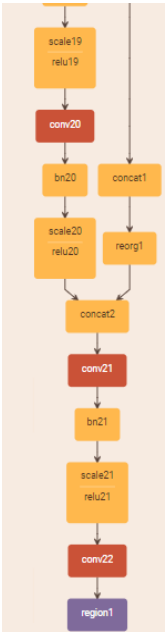
b. A new layer is added for that purpose: reorg



F1.5: Multi-Scale Training

- Remove FC layers: Can accept any size of inputs, enhance model robustness.
- Size across 320, 352, …, 608. Change per 10 epochs
  [border % 32 = 0, decided by down sampling]





II. One Stage Detection

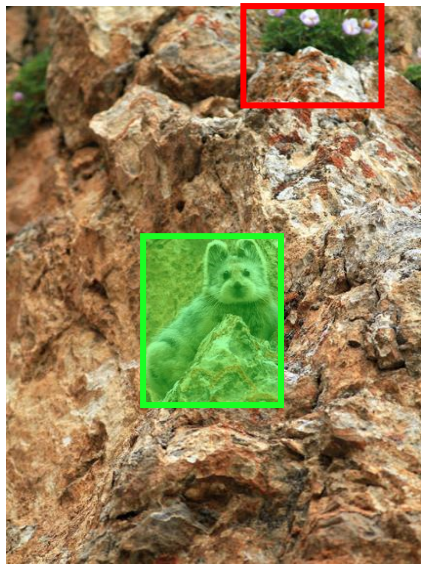
F. Yolo V2 [2016, Joseph]

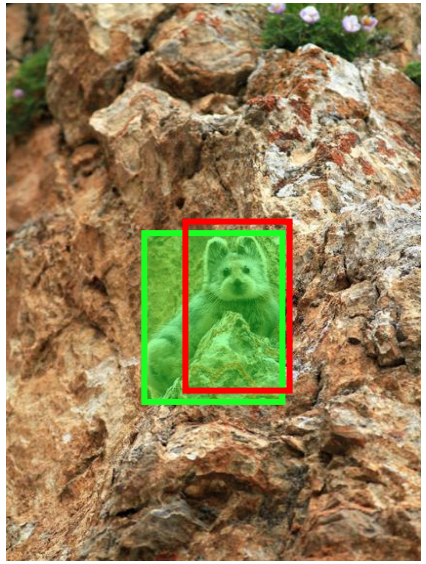
F2. Anchor in Yolo V2

F2.0: What is Anchor? Why anchor?

a. Pre-defined virtual bboxes

b. Final bboxes are generated from them





II. One Stage Detection

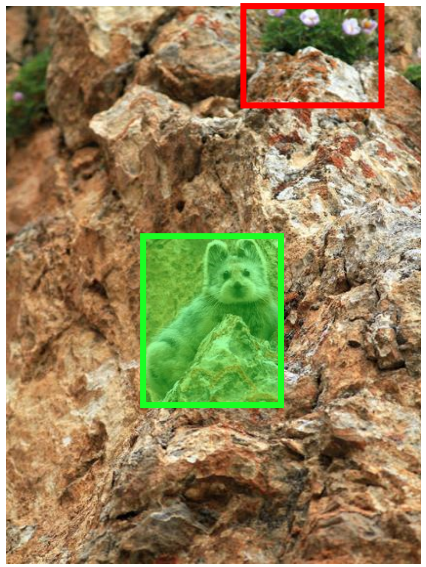
F. Yolo V2 [2016, Joseph]

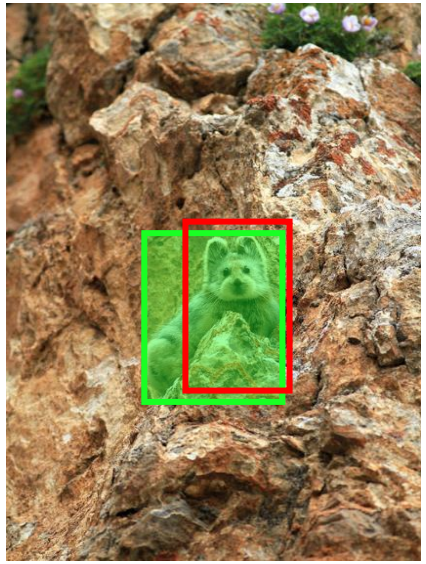
F2. Anchor in Yolo V2

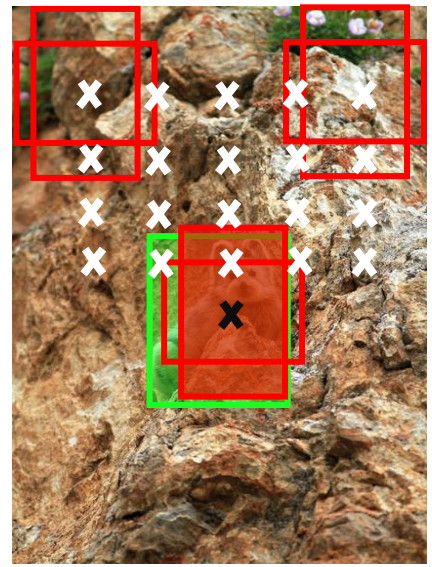
F2.0: What is Anchor? Why anchor?

a. Pre-defined virtual bboxes

b. Final bboxes are generated from them





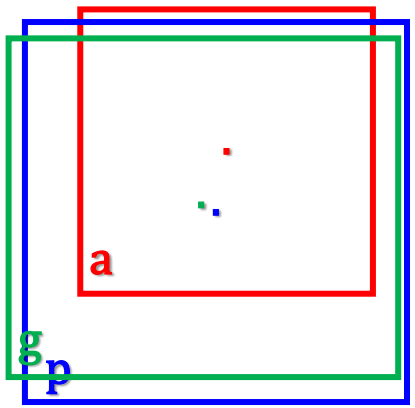


II. YOLO v2

B. Anchor与loss function

F2. Anchor in Yolo V2

F2.1: Anchors? Ground Truth BBoxes? Prediction?



We hope **a** -> **g**,
but actually, **a** generates **p**.

So as long as **p** -> **g**,
We get a good result.

So as long as the offset of
p - **a** -> **g** – **a**
We get a good result

So here we hope $t_\cdot^p \rightarrow t_\cdot^g$

predicated bbox: x_p, y_p, w_p, h_p

anchor bbox: x_a, y_a, w_a, h_a

ground truth bbox: x_g, y_g, w_g, h_g

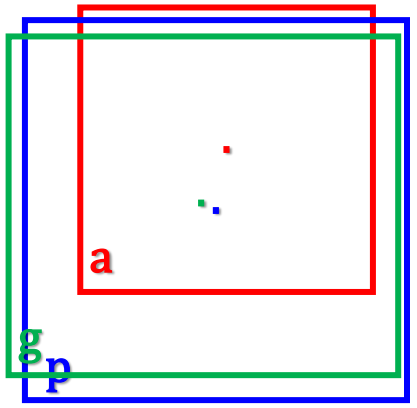
$$f(x) = \begin{cases} t_x^p = \frac{x_p - x_a}{w_a}, t_y^p = \frac{y_p - y_a}{h_a} \\ t_w^p = log\left(\frac{w_p}{w_a}\right), t_h^p = log\left(\frac{h_p}{h_a}\right) \\ t_x^g = \frac{x_g - x_a}{w_a}, t_y^g = \frac{y_g - y_a}{h_a} \\ t_w^g = log\left(\frac{w_g}{w_a}\right), t_h^g = log\left(\frac{h_g}{h_a}\right) \end{cases}$$

II. YOLO v2

B. Anchor与loss function

F2. Anchor in Yolo V2

F2.1: Anchors? Ground Truth BBoxes? Prediction?



We hope a -> g,
but actually a generates p.

So as long as p ->g,
We get a good result.

So as long as the offset of
p - a -> g – a
We get a good result

So here we hope $t^p \rightarrow t^g$

predicated bbox: x_p, y_p, w_p, h_p

anchor bbox: x_a, y_a, w_a, h_a

ground truth bbox: x_g, y_g, w_g, h_g

$$f(x) = \begin{cases} t_x^p = \frac{x_p - x_a}{w_a}, t_y^p = \frac{y_p - y_a}{h_a} \\ t_w^p = log\left(\frac{w_p}{w_a}\right), t_h^p = log\left(\frac{h_p}{h_a}\right) \\ t_x^g = \frac{x_g - x_a}{w_a}, t_y^g = \frac{y_g - y_a}{h_a} \\ t_w^g = log\left(\frac{w_g}{w_a}\right), t_h^g = log\left(\frac{h_g}{h_a}\right) \end{cases}$$

Target: -> [regress offset]

: generated directly **FROM NET**

: calculated [Encode]

: calculated – final outputs [Decode]

II. One Stage Detection

F. Yolo V2 [2016, Joseph]

F2. Anchor in Yolo V2

F2.2: Anchor size and number

a. Faster RCNN: 9 by hands

b. Yolo: 5 by K-Means [dist: 1 – iou(bbox, cluster)]

F2.3: Anchors, Truth BBoxes & Predicted BBoxes

- Anchors: 0.57273, 0.677385, …, 9.77052, 9.16828 [10 numbers]

$$a_{w_0}, \quad a_{h_0} \quad , \ldots, \quad a_{w_4}, \quad a_{h_4}$$

$$\text{anchors}[0] = a_{w_i} = \frac{a_{w_{ori}}}{W} * 13$$ [not strict, just aiming to say how we get those numbers]

- Truth Bboxes:

II. One Stage Detection

F. Yolo V2 [2016, Joseph]

F2. Anchor in Yolo V2

F2.3: Anchors, Truth BBoxes & Predicted BBoxes

- **Anchors:** $\text{anchors}[0] = a_{w_i} = \frac{a_{w_{ori}}}{W} * 13$ [not strict, just aiming to say how we get those numbers]
- **Truth Bboxes:**

original bbox: $[x_o, y_o, w_o, h_o] \epsilon [0, W|H]$

↓ normalize in 0~1

normalized original bbox: $[x_r, y_r, w_r, h_r] \epsilon [0,1]$

$[x_r, y_r, w_r, h_r] = [x_o/W, y_o/H, w_o/W, h_o/H]$

↓ transfer to feature map size: 13 x 13

box: $[x, y, w, h] \epsilon (0,13]$

$[x, y, w, h] = [x_r, y_r, w_r, h_r] * (13|13)$

↓ save this for calculating

↓ transfer to 0~1 corresponding to each grid cell

final box: $[x_f, y_f, w_f, h_f] \epsilon [0,1]$

↓ save this for calculating

↓ transfer to 0~1 corresponding to each grid cell

final box: $[x_f, y_f, w_f, h_f] \epsilon [0,1]$

$$\begin{cases} x_f = x - i \\ y_f = y - j \\ w_f = \log(w/anchors[0]) \\ h_f = \log(h/anchors[1]) \end{cases}$$

II. One Stage Detection

F. Yolo V2 [2016, Joseph]

F2. Anchor in Yolo V2

F2.3: Anchors, Truth BBoxes & Predicted BBoxes

$$f(x) = \begin{cases} t_x^p = \frac{x_p - x_a}{w_a}, t_y^p = \frac{y_p - y_a}{h_a} \\ t_w^p = log\left(\frac{w_p}{w_a}\right), t_h^p = log\left(\frac{h_p}{h_a}\right) \\ t_x^g = \frac{x_g - x_a}{w_a}, t_y^g = \frac{y_g - y_a}{h_a} \\ t_w^g = log\left(\frac{w_g}{w_a}\right), t_h^g = log\left(\frac{h_g}{h_a}\right) \end{cases}$$

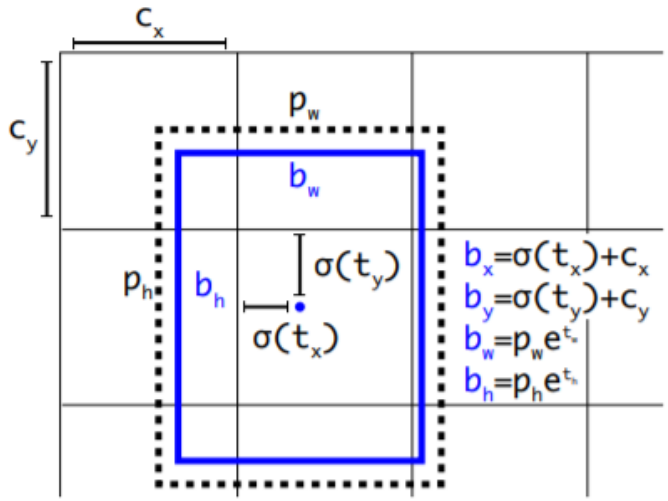


Figure 3: Bounding boxes with dimension priors and location prediction. We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function.

$$\begin{cases} \sigma(t_x^p) = b_x - C_x, \sigma(t_y^p) = b_y - C_y \\ t_w^p = log\left(\frac{b_w}{p_w}\right), t_h^p = log\left(\frac{b_h}{p_h}\right) \\ t_x^g = g_x - g_{x.floor()}, t_y^g = g_y - g_{y.floor()} \\ t_w^g = log\left(\frac{g_w}{p_w}\right), t_h^g = log\left(\frac{g_h}{p_h}\right) \end{cases}$$

II. One Stage Detection

F. Yolo V2 [2016, Joseph]

F2. Anchor in Yolo V2

F2.3: Anchors, Truth BBoxes & Predicted BBoxes

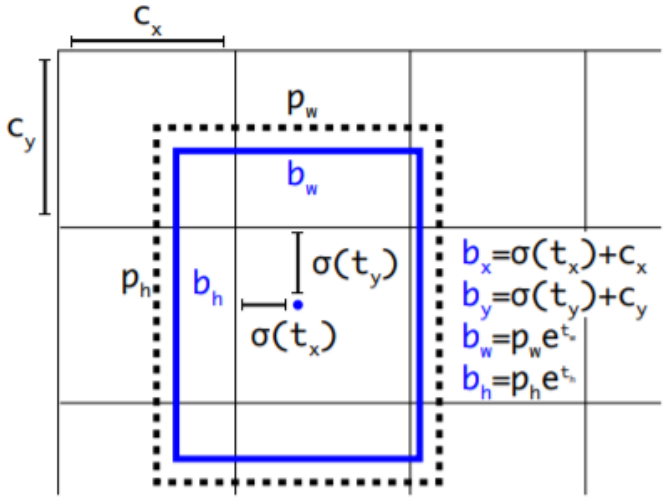


Figure 3: Bounding boxes with dimension priors and location prediction. We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function.

Anchors: $\text{anchors}[0] = a_{w_i} = \frac{a_{w_{ori}}}{W} * 13$ [not strict, just aiming to say how we get those numbers]

Truth Bboxes:

original bbox: $[x_o, y_o, w_o, h_o] \epsilon [0, W|H]$

↓ normalize in 0~1

normalized original bbox: $[x_r, y_r, w_r, h_r] \epsilon [0,1]$

$[x_r, y_r, w_r, h_r] = [x_o/W, y_o/H, w_o/W, h_o/H]$

↓ transfer to feature map size: 13 x 13

box: $[x, y, w, h] \epsilon (0,13]$

$[x, y, w, h] = [x_r, y_r, w_r, h_r] * (13|13)$

↓ save this for calculating

↓ transfer to 0~1 corresponding to each grid cell

final box: $[x_f, y_f, w_f, h_f] \epsilon [0,1]$

↓ save this for calculating

↓ transfer to 0~1 corresponding to each grid cell

final box: $[x_f, y_f, w_f, h_f] \epsilon [0,1]$

$$\begin{cases} x_f = x - i \\ y_f = y - j \\ w_f = \log(w/anchors[0]) \\ h_f = \log(h/anchors[1]) \end{cases}$$

$$\begin{cases} \sigma(t_x^p) = b_x - C_x, \sigma(t_y^p) = b_y - C_y \\ t_w^p = log\left(\frac{b_w}{p_w}\right), t_h^p = log\left(\frac{b_h}{p_h}\right) \\ t_x^g = g_x - g_{x.floor}, t_y^g = g_y - g_{y.floor} \\ t_w^g = log\left(\frac{g_w}{p_w}\right), t_h^g = log\left(\frac{g_h}{p_h}\right) \end{cases}$$

$$f(x) = \begin{cases} t_x^p = \frac{x_p - x_a}{w_a}, t_y^p = \frac{y_p - y_a}{h_a} \\ t_w^p = log\left(\frac{w_p}{w_a}\right), t_h^p = log\left(\frac{h_p}{h_a}\right) \\ t_x^g = \frac{x_g - x_a}{w_a}, t_y^g = \frac{y_g - y_a}{h_a} \\ t_w^g = log\left(\frac{w_g}{w_a}\right), t_h^g = log\left(\frac{h_g}{h_a}\right) \end{cases}$$

$$\begin{cases} \sigma(\cdot) \in (0,1) \quad b. \in (0,13) \quad (C_x, C_y) \leftrightarrow (i,j) \leftrightarrow grid\ ID \\ p_w \leftrightarrow p_h \in (0,13) \\ t_x^g \leftrightarrow t_y^g \in (0,1) \quad g_x \leftrightarrow g_y \in (0,13) \quad (g_{x.floor}, g_{x.floor}) \leftrightarrow (C_x, C_y) \\ g_w \leftrightarrow g_h \in (0,13) \end{cases}$$

II. One Stage Detection

F. Yolo V2 [2016, Joseph]

F3. Problems

- Better for small & crowded objects.

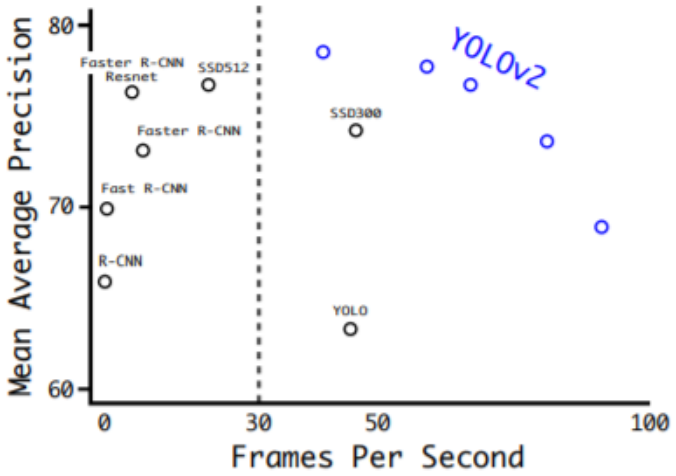


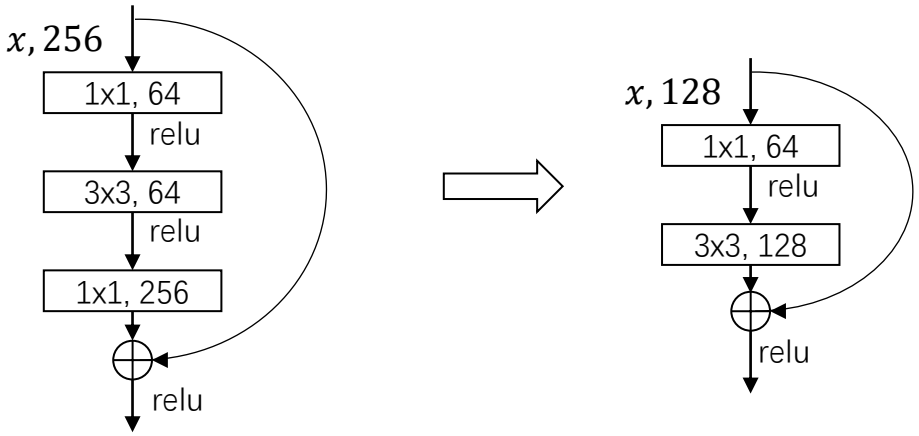
Figure 4: Accuracy and speed on VOC 2007.

II. One Stage Detection

G. Yolo V3 [2018, Joseph]

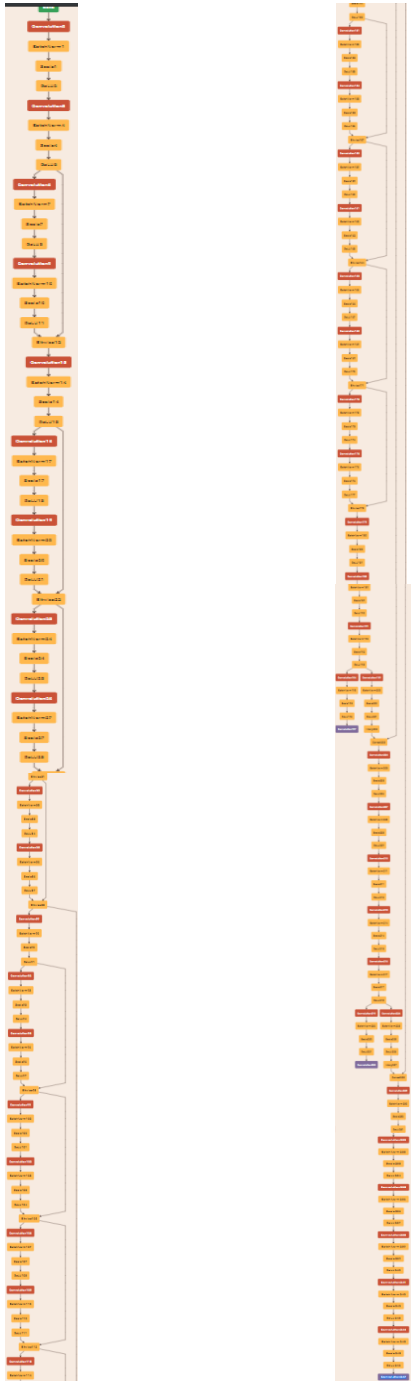
G1. Improvements

G1.1: New structure



G1.2: Multiscale Structure

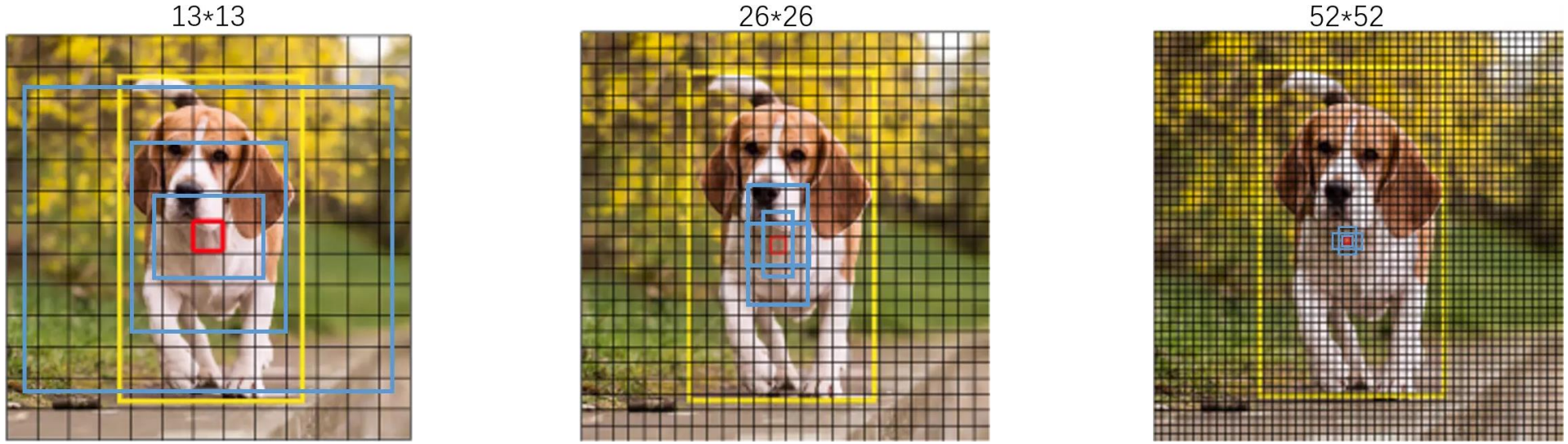
- 3 scales; 3 anchors per scale per grid
- /32, small scale (13 x 13) —> large anchor
- /16, mid scale (26 x 26) —> medium anchor
- /8, large scale (52 x 52) —> small anchor



II. One Stage Detection

G. Yolo V3 [2018, Joseph]

G1. Improvements



G1.3: Change Classification :

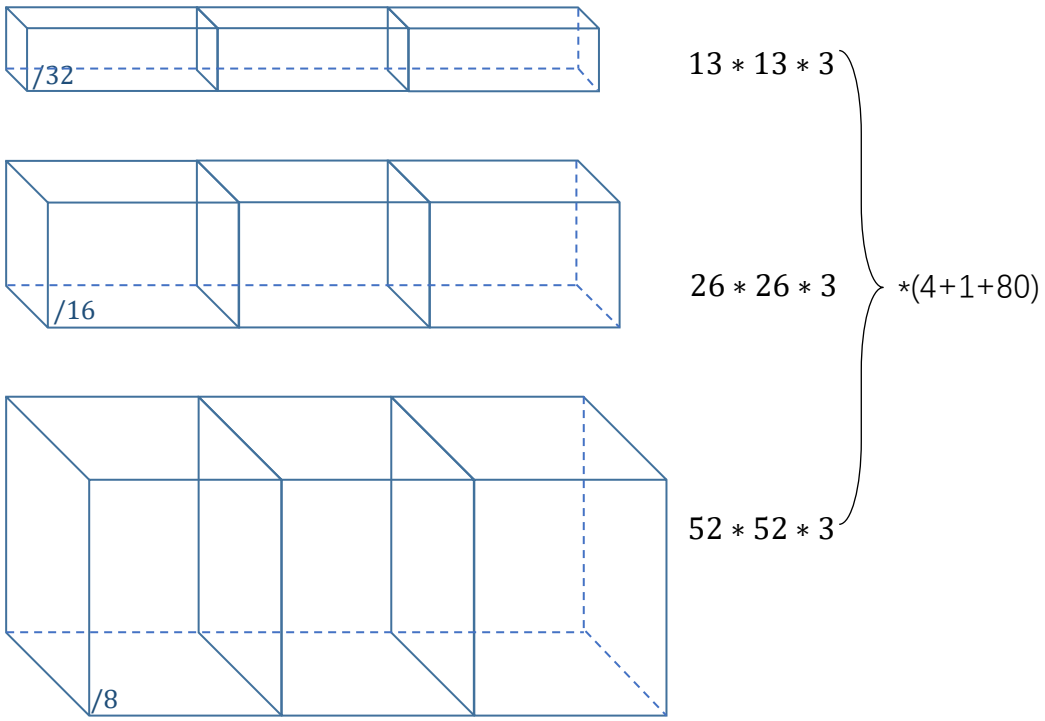
- 80 classes, from softmax —> logistic

II. One Stage Detection

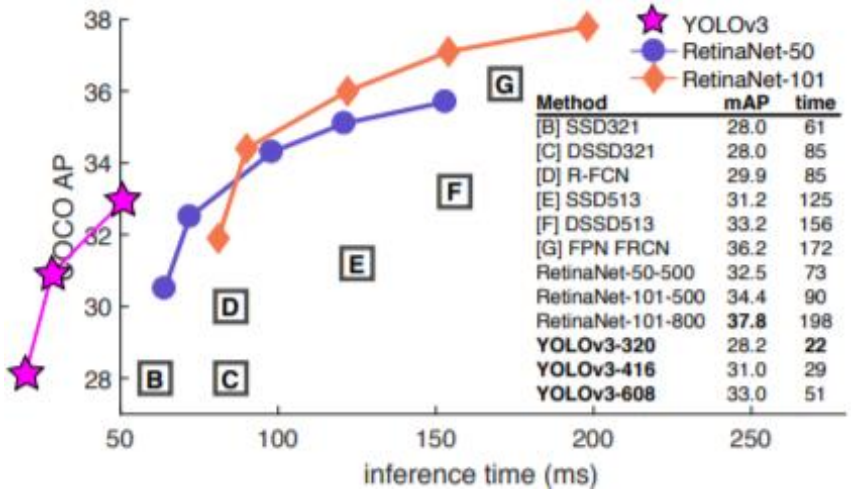
G. Yolo V3 [2018, Joseph]

G2. Summary

G2.1: Output



F2.2: Performance

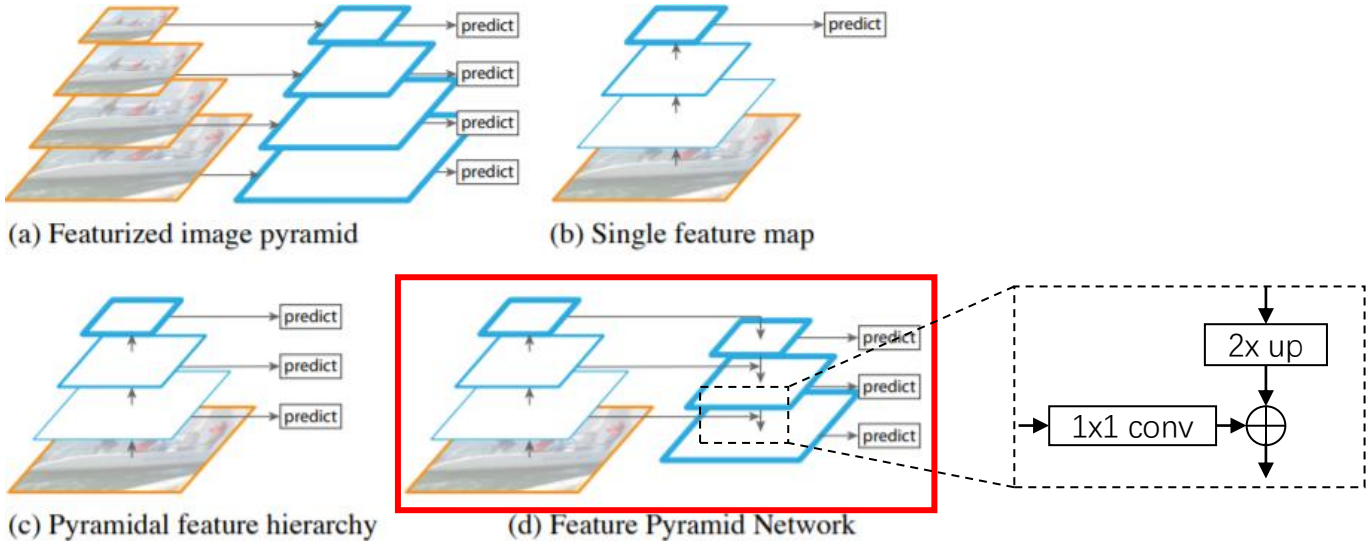


Method	mAP	time
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

II. One Stage Detection

G. Yolo V3 [2018, Joseph]

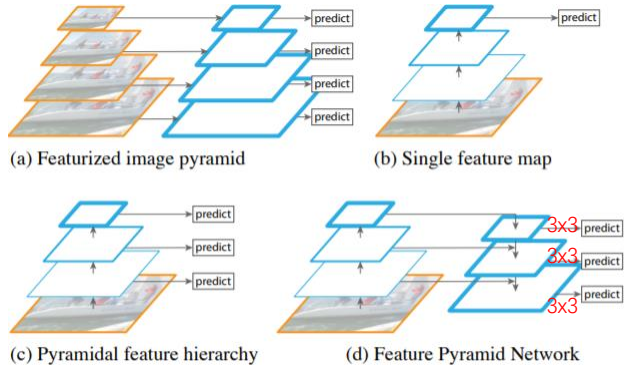
G3. FPN Net [2017, Lin]



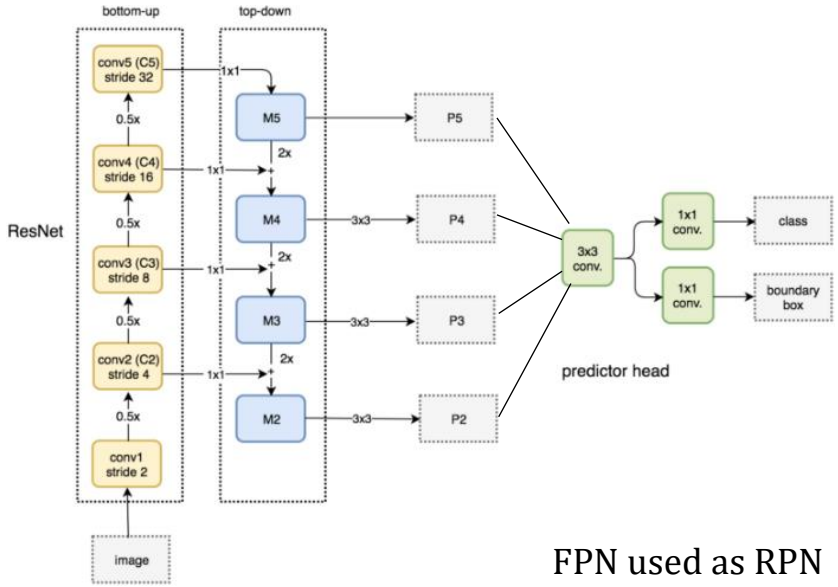
II. One Stage Detection

G. Yolo V3 [2018, Joseph]

G3. FPN Net [2017, Lin]



- **Pros:**
  1. Lower layers have accurate localization info;
     Higher layers have ample semantic info
     FPN combine them together. Like U-Net
  2. Feature detector: worked as a part in a whole bigger network.
- **Details:**
  1. 3x3convolution is appended on each merged map to generate the final feature map to reduce the aliasing effect
  2. Feature dimension at output is 256.
  3. When used in FPN. P2-P6. Anchor areas: $32^2, \sim, 512^2$ per scale, with ratio {1:2,1:1,2:1}, 15 in total. And, predictor head is shared.



FPN used as RPN

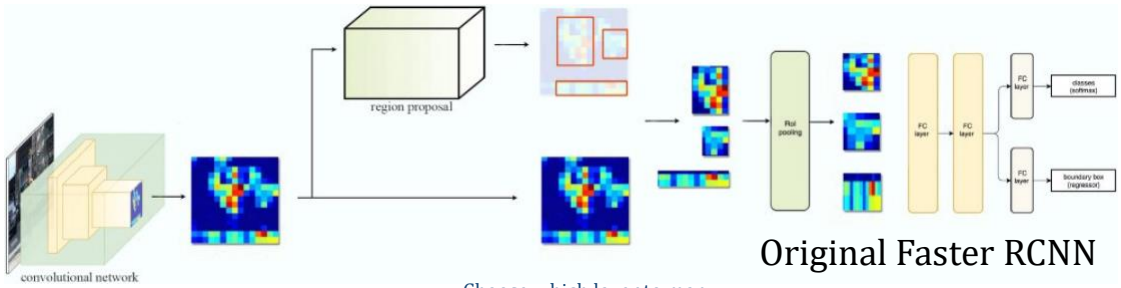
II. One Stage Detection

G. Yolo V3 [2018, Joseph]

G3. FPN Net [2017, Lin]

- **Details:**

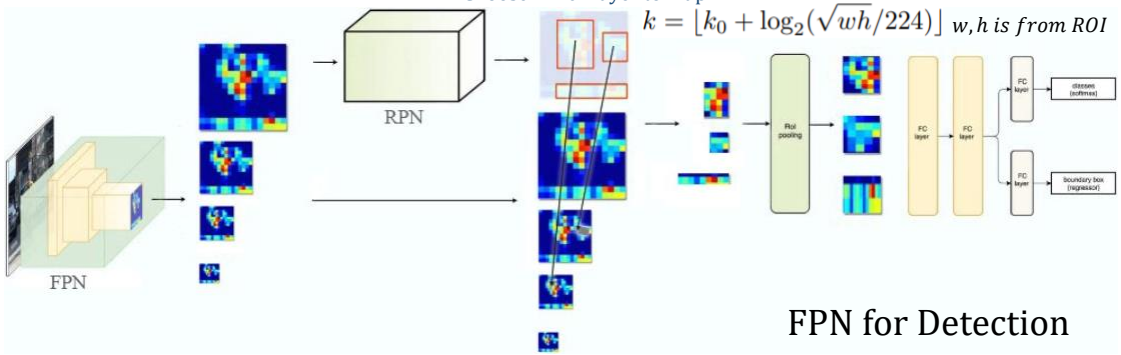
  4. When used in faster rcnn:



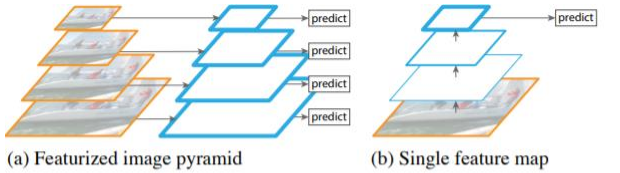
Original Faster RCNN

Choose which layer to map

$k = \lfloor k_0 + \log_2(\sqrt{wh}/224) \rfloor$ *w,h is from ROI*

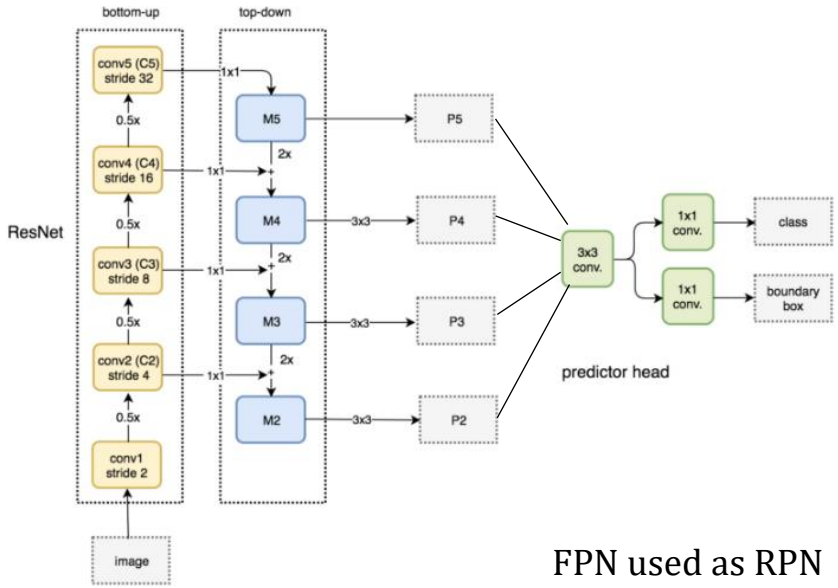


FPN for Detection



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

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



FPN used as RPN

II. One Stage Detection

H. Yolo V4 [2020, Alexey]

- **Improvements:**

1. Data Augmentation:

Mosaic

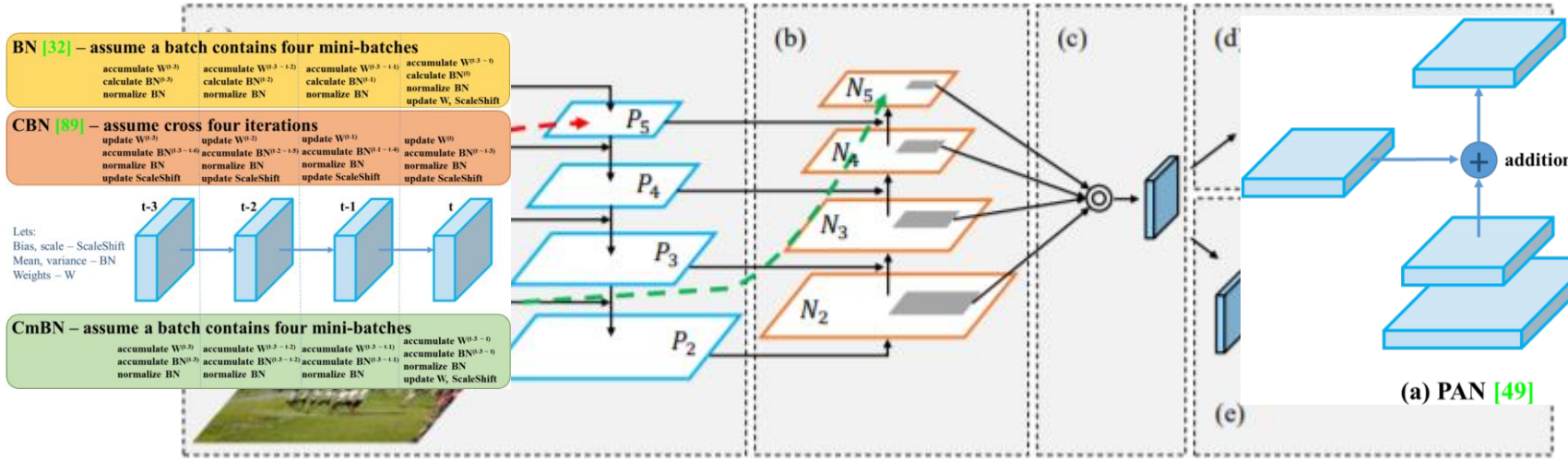
2. Data Augmentation[cvpr2017, human pose]:

Self-Adversarial Training

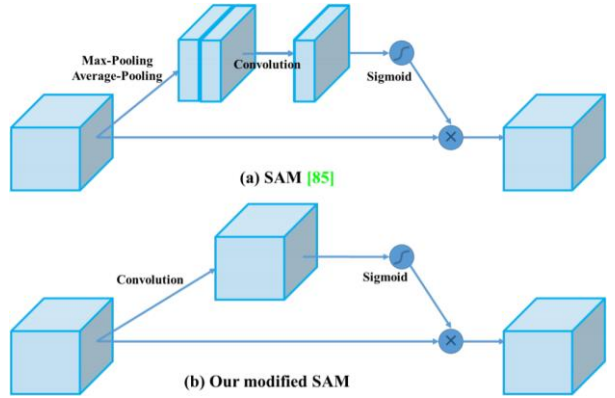
a. alter input

b. train on altered inputs

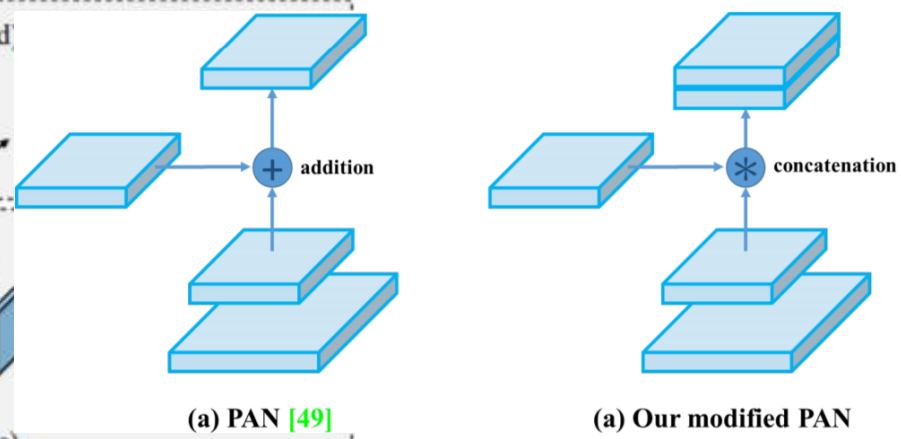
3. CBN —> CmBN



4. Modified SAM:



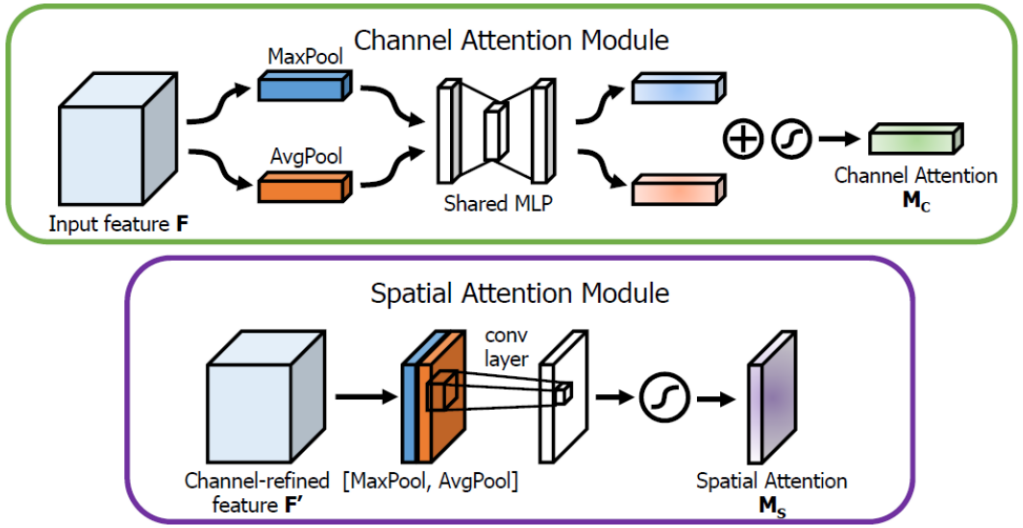
5. Modified PAN:

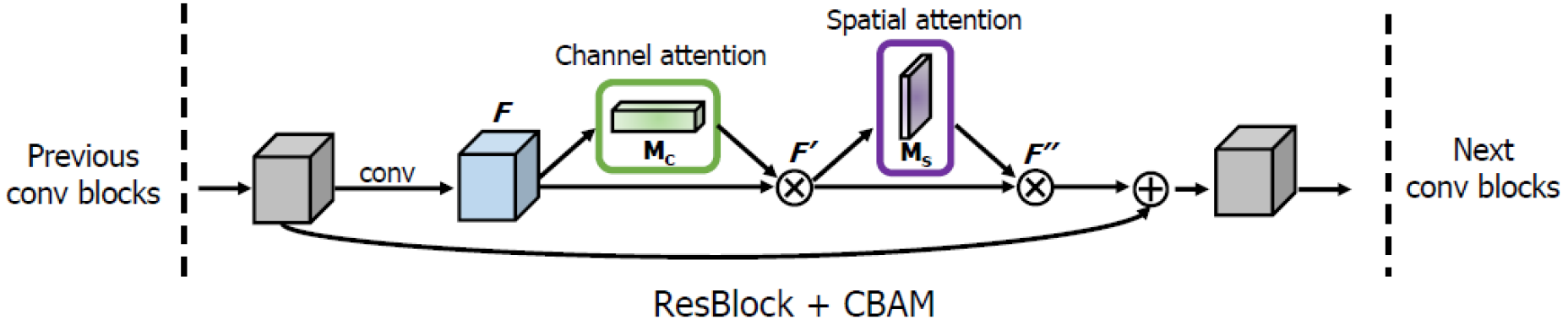


II. One Stage Detection

H. Yolo V4 [2020, Alexey]

- **Attention Module:**

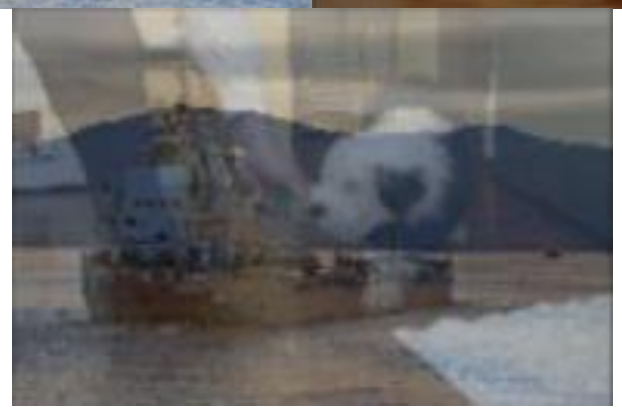




II. One Stage Detection

H. Yolo V4 [2020, Alexey]

- **Data augmentation:**
  CutMix = Mixup + Cutout

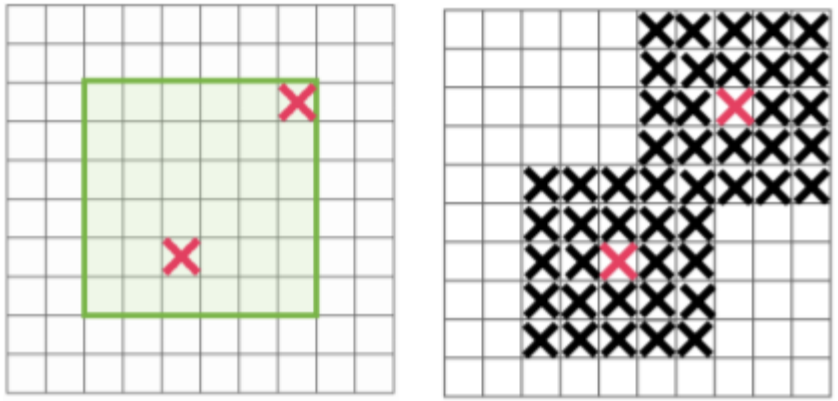


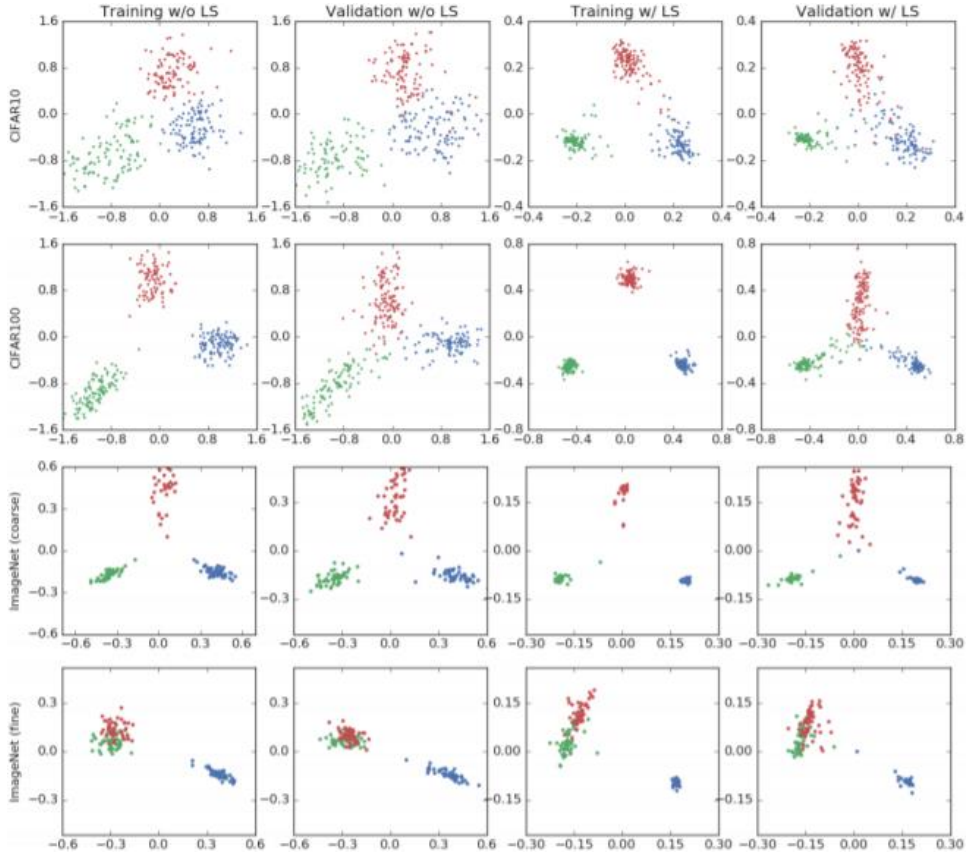
II. One Stage Detection

H. Yolo V4 [2020, Alexey]

- **Regularization:**

  1. Label smoothing / 2. Dropblock





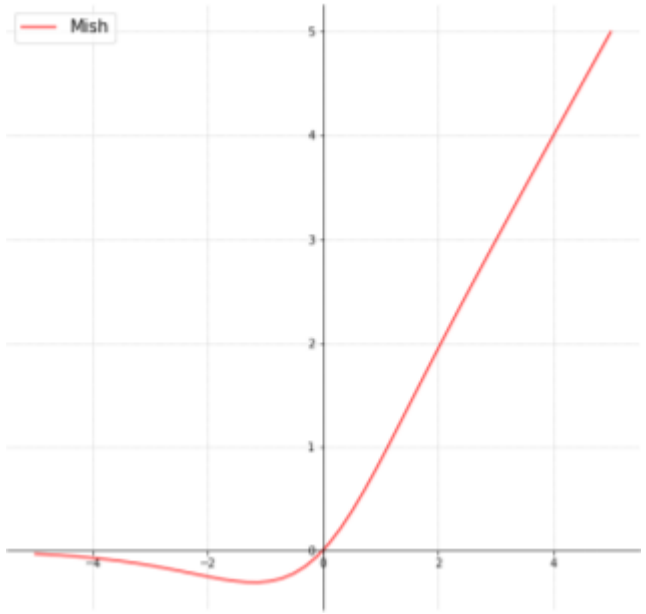
II. One Stage Detection

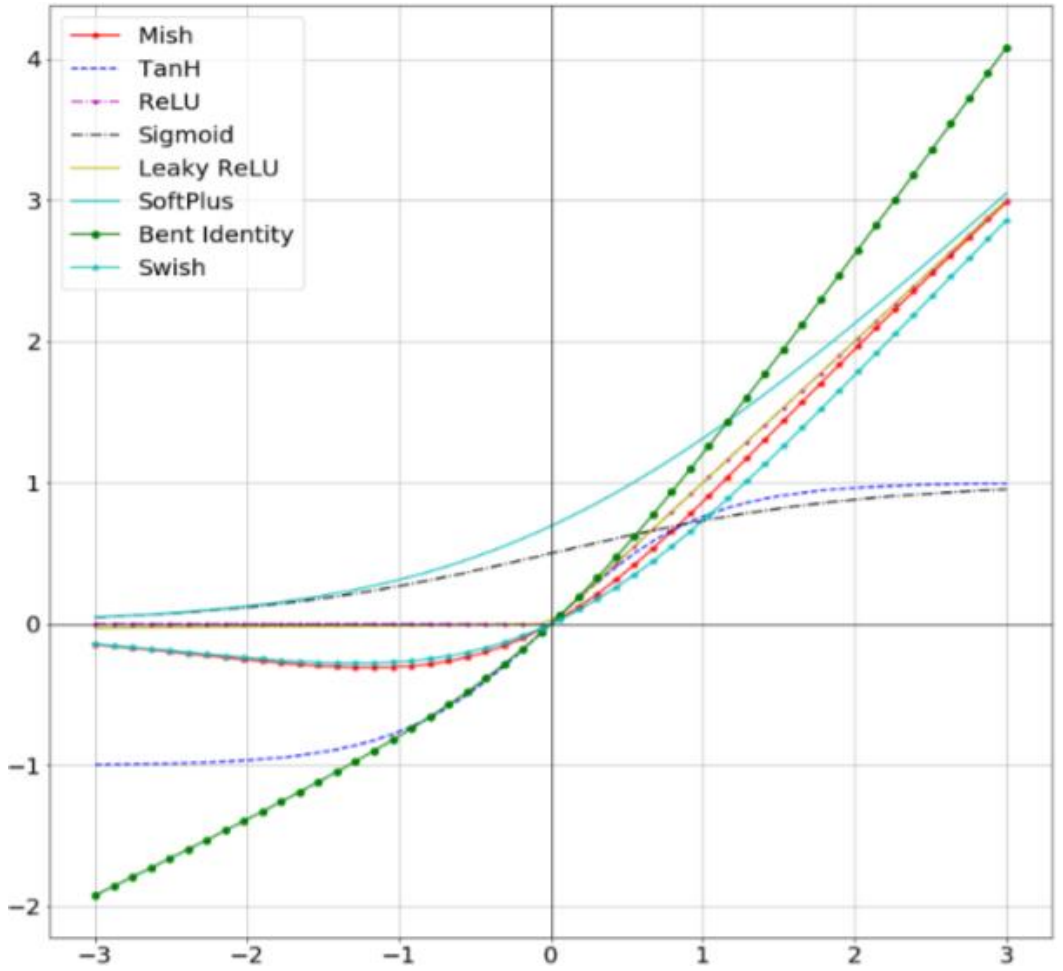
H. Yolo V4 [2020, Alexey]

- **Activation Function:**

1. Mish

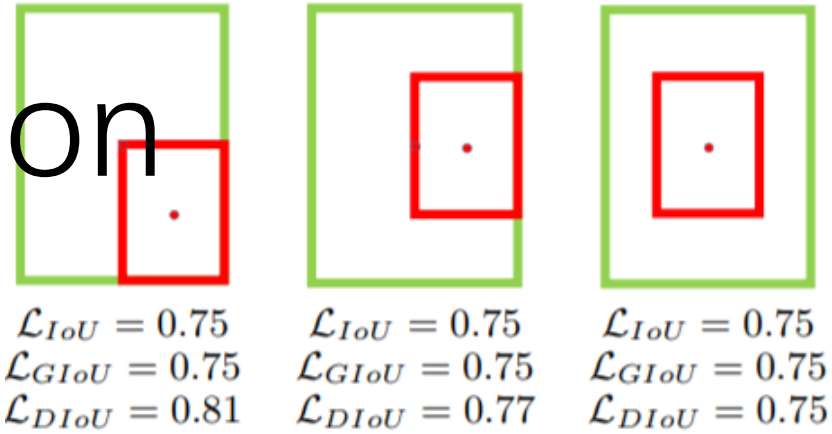
$$f(x) = x \cdot \tanh(\varsigma(x))$$
$$\varsigma(x) = \ln(1 + e^x) — softplus$$





II. One Stage Detection

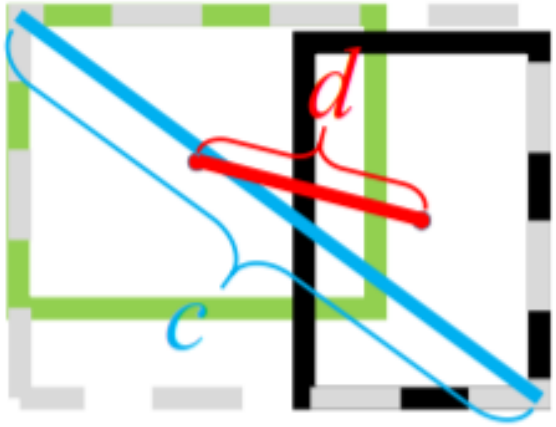
H. Yolo V4 [2020, Alexey]



- **Loss Function:**

1. CIoU Loss

$IoU \rightarrow GIoU \rightarrow DIoU \rightarrow CIoU$

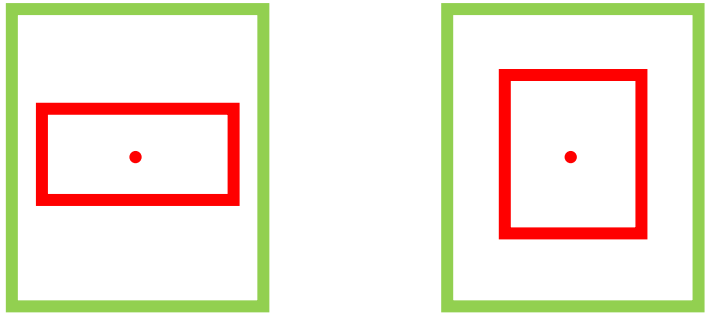


3 keypoints when using IoU-based loss:

a. IoU – Overlap area [IoU Loss]

b. Central point distance [GIoU, DIoU]

c. Aspect ratio [CIoU]



II. One Stage Detection

I. RetinaNet [2018, Lin]

- **Structure:**

  Resnet + FPN + FCN :

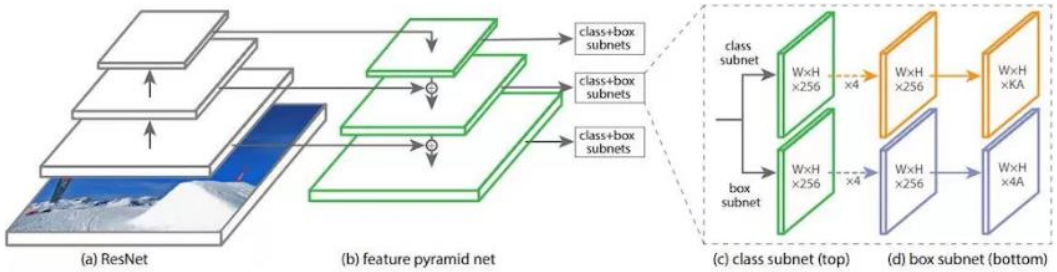


Figure 3. The one-stage **RetinaNet** network architecture uses a Feature Pyramid Network (FPN) [20] backbone on top of a feedforward ResNet architecture [16] (a) to generate a rich, multi-scale convolutional feature pyramid (b). To this backbone RetinaNet attaches two subnetworks, one for classifying 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.

- **Focal Loss:**

  Q. Why one-stage performs worse than two stage?

  A. 1. Because neg/pos samples are extremely unbalanced

  2. Gradient is dominated by easy samples.

  S. We can use FOCAL LOSS to solve it.

II. One Stage Detection

I. RetinaNet [2018, Lin]

➢ **Structure:**

Resnet + FPN + FCN :

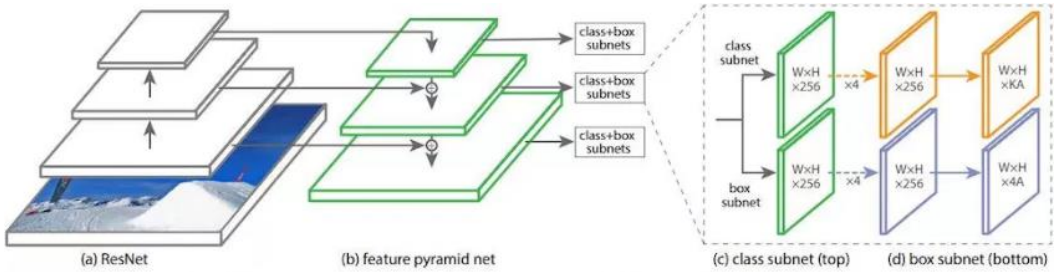


Figure 3. The one-stage **RetinaNet** network architecture uses a Feature Pyramid Network (FPN) [20] backbone on top of a feedforward ResNet architecture [16] (a) to generate a rich, multi-scale convolutional feature pyramid (b). To this backbone RetinaNet attaches two subnetworks, one for classifying 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.

➢ **Focal Loss:**

Q. Why one-stage performs worse than two stage?

A. 1. Because neg/pos samples are extremely unbalanced

2. Gradient is dominated by easy samples.

S. We can use FOCAL LOSS to solve it.

✓ **Original cross entropy:**

$$CE = -\log(P_t) = \begin{cases} -log\hat{y}, & y = 1 \\ -\log(1-\hat{y}), & y = 0 \end{cases}$$

✓ **Solution to unbalanced samples:**

$$CE = -\alpha_t \log(P_t)$$

—— $different\ weight\ for\ pos/neg\ samples$

✓ **Solution to dominated samples:**

$$CE = -(1-P_t)^{\gamma} \log(P_t)$$

—— $P_t \to 0, (1-P_t) \to 1, learn\ normally$

$P_t \to 1, (1-P_t) \to 0, learn\ nothing$

✓ **Focal loss:**

$$CE = -\alpha_t(1-P_t)^{\gamma} \log(P_t)$$

$$= \begin{cases} -\alpha(1-\hat{y})^{\gamma} log\hat{y}, & y = 1 \\ -(1-\alpha)\hat{y}^{\gamma} \log(1-\hat{y}), & y = 0 \end{cases}$$

II. One Stage Detection

J. Some Resources

- I1: Yolo V1
- I2: Yolo V2
- I3: Yolo V3 / From scratch
- I4: Yolo v4 ideas I
  Yolo v4 ideas II

How to implement a YOLO (v3) object detector from scratch in PyTorch: Part 1

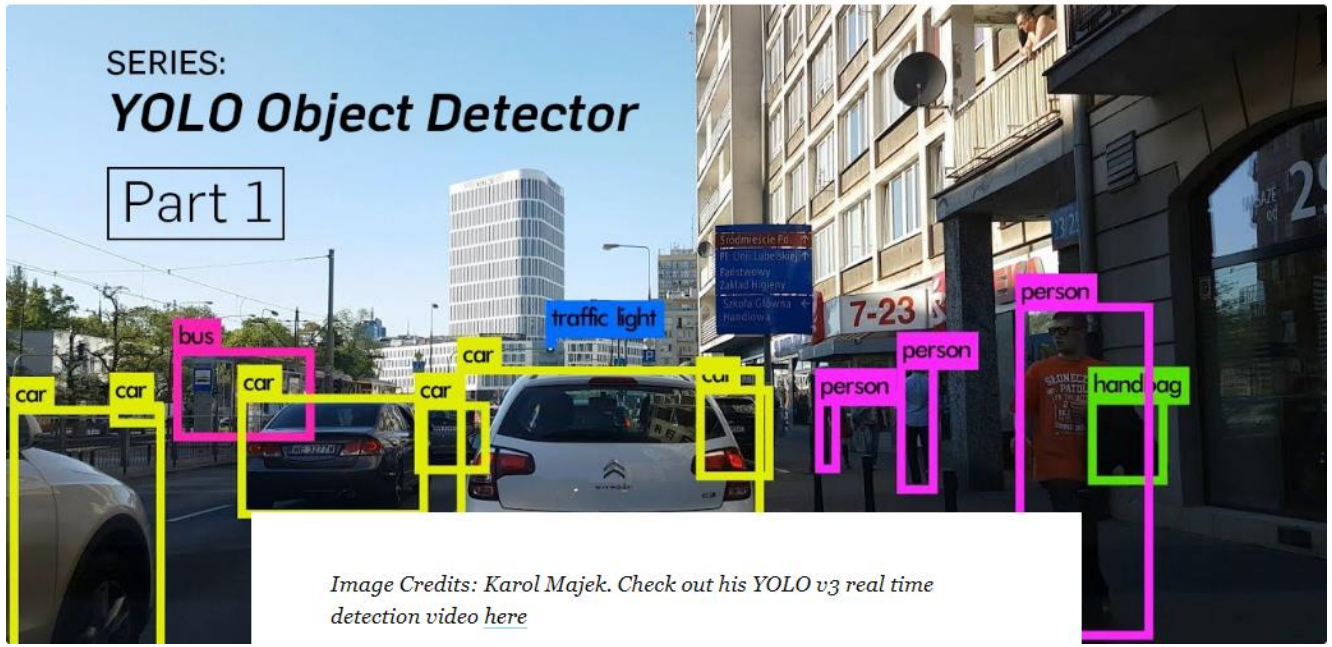
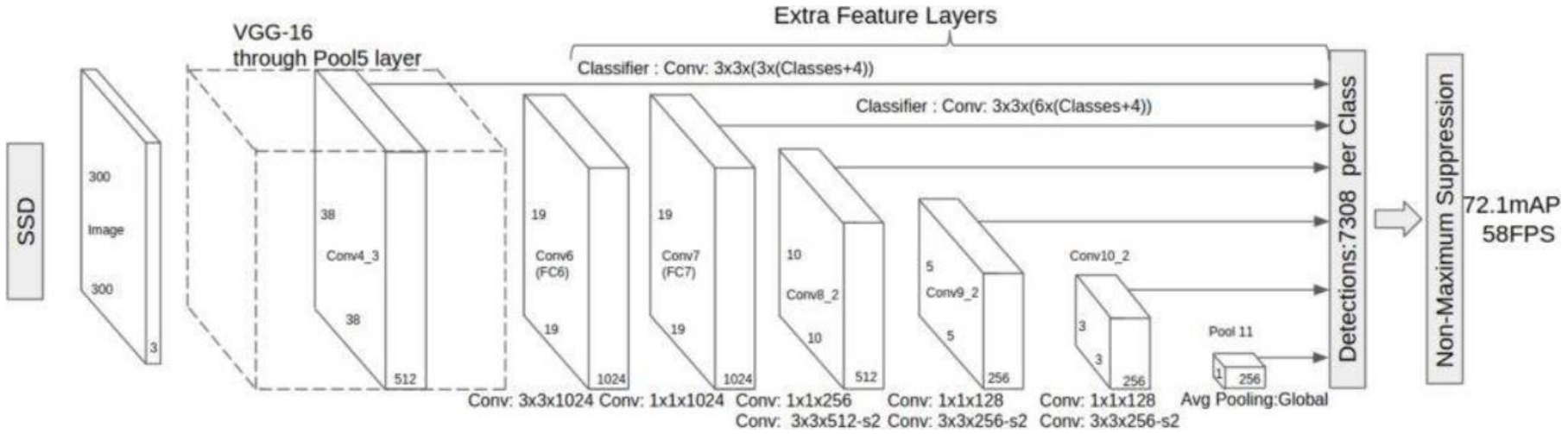


Image Credits: Karol Majek. Check out his YOLO v3 real time detection video here

III. Other Methods

K. Other Method

- SSD Series: (2015, Liu, another one-stage, also popular!)
  - **Methods based on SSD**
  - **FaceBoxes / Github (original PyTorch version)/**
    Exp: Training part (Python, landmarks det, new training method, new network structure, new NMS, etc.);
    Reference part (C++, acceleration, etc.)

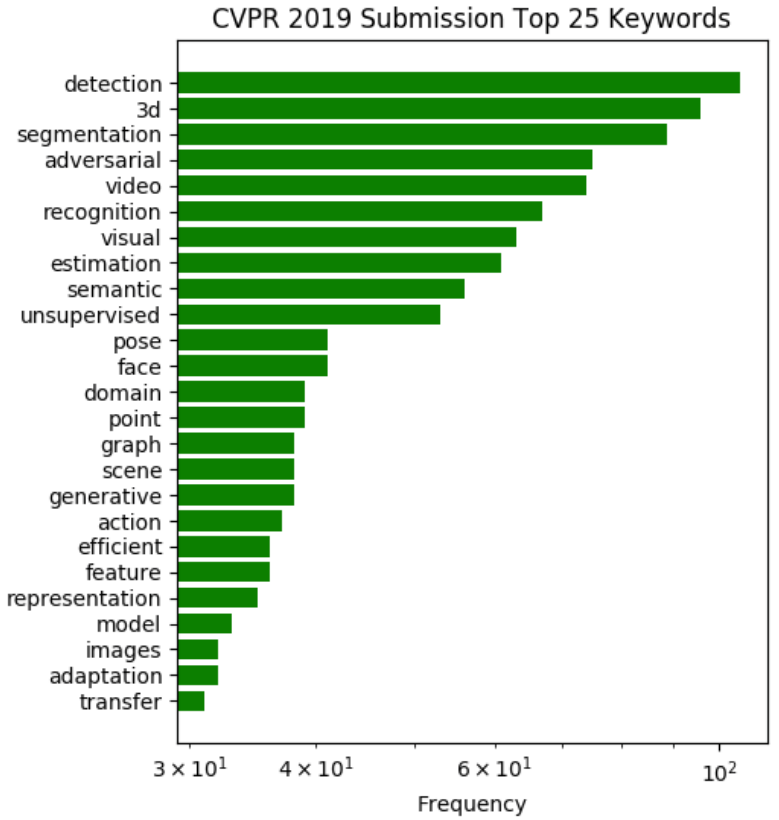


III. Anchor Free Methods

III. Other Methods

L. Trend

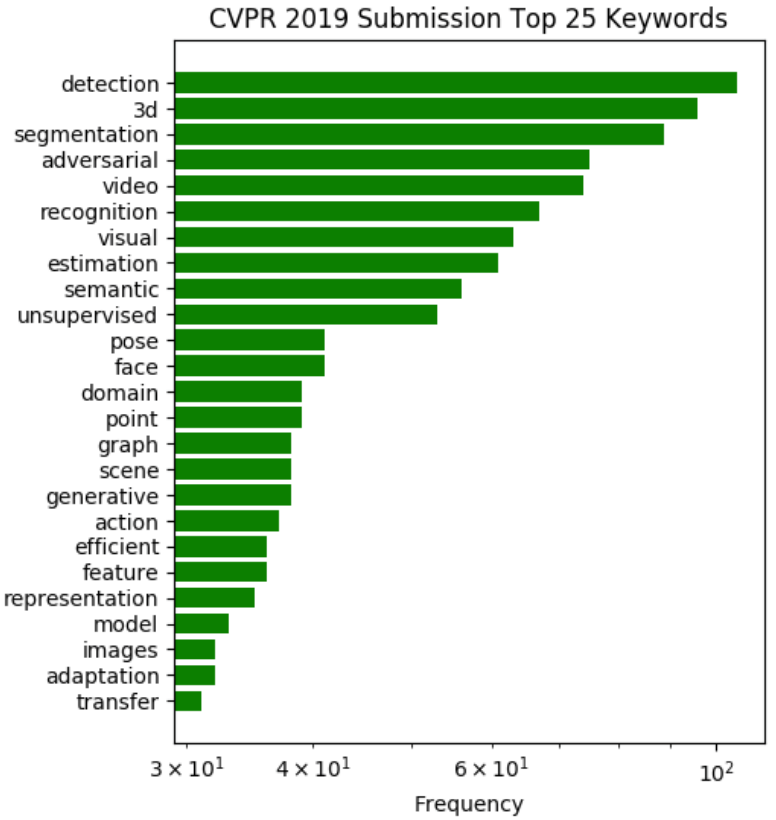
- Anchor free net is a trend
  - Lots of hyperparameters: sizes, aspect-ratios, number of anchors, ……
  - Hard to generalize: different datasets have different data shape, need to redesign
  - Difficult to train: unbalanced positive / negative samples
  - Complex calculation
  - Myriads of redundancy



III. Other Methods

L. Trend

- Anchor free net is a trend
  - CornerNet: 244, 03/2019 || CornerNet-Lite: 18, 04/2019
  - FoveaBox: 29, 04/2019
  - CenterNet: Objects as Points -78, 04/2019
    ~~Keypoint Triplets for Object Detection~~
  - FCOS: 66, 08/2019

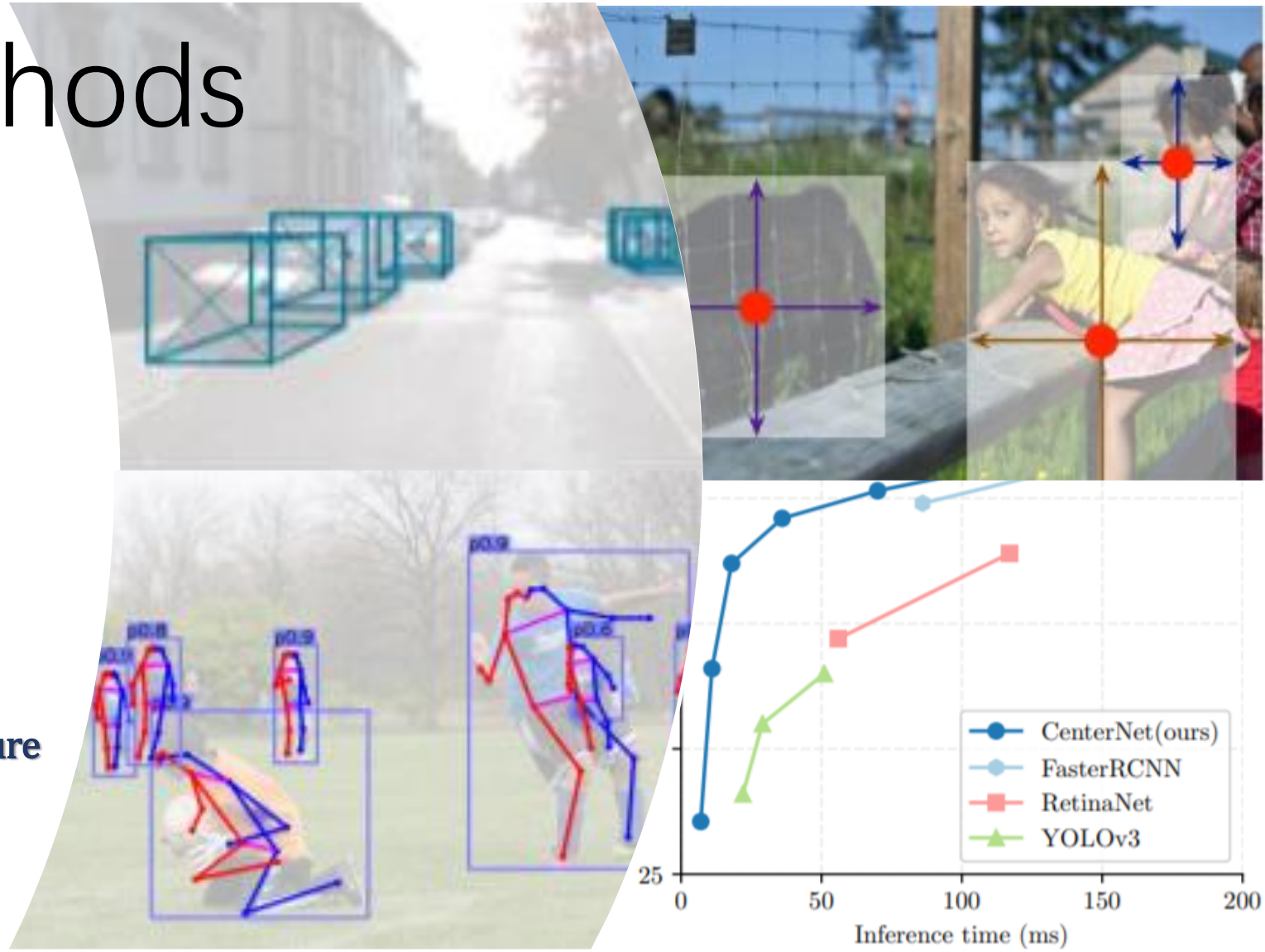


III. Other Methods

M. CenterNet [2019, Zhou]

➢ Features:

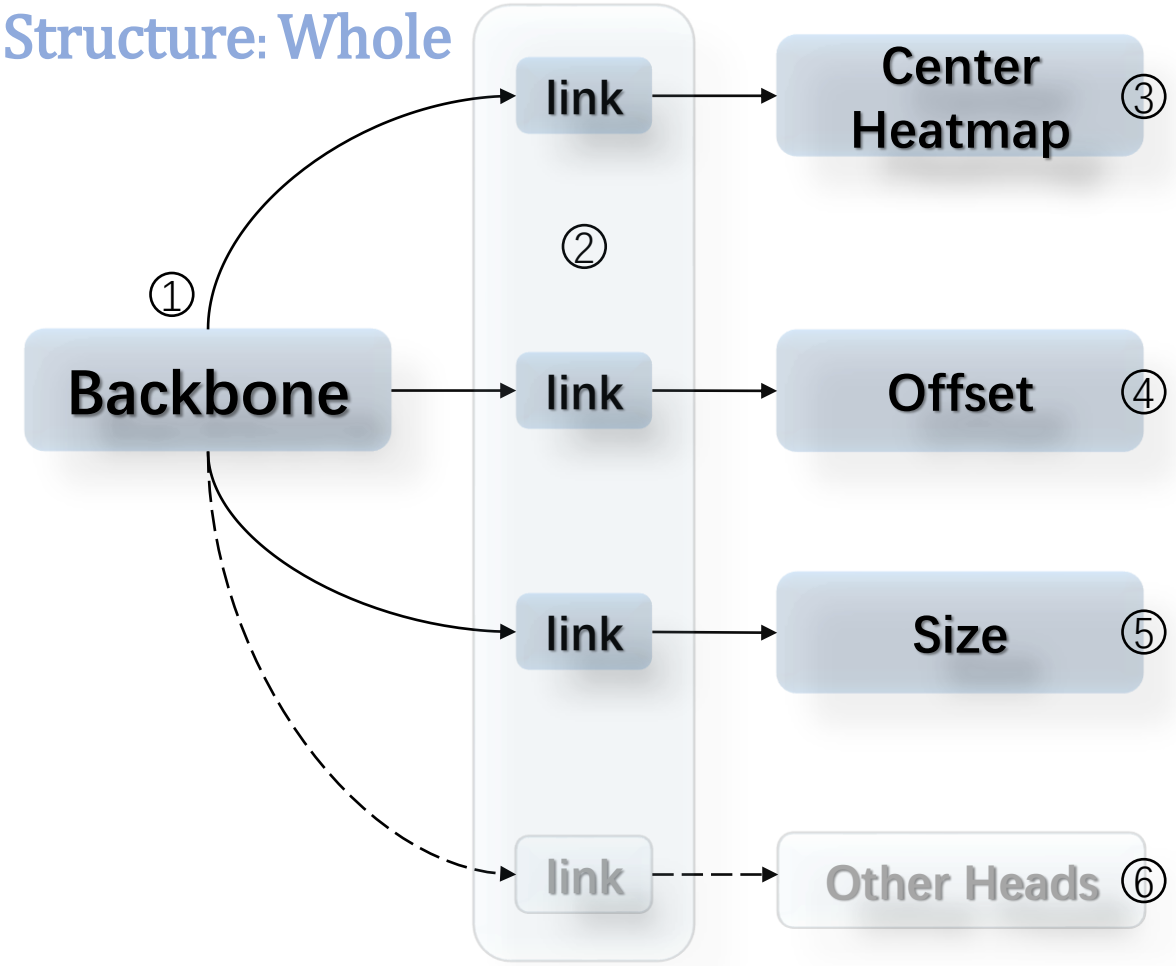
- More accurate & faster
- Detect "objects as points"
  [only detect center points]
- Multiple functions in one structure
- No need for post-processing
  [NMS etc.]



III. Other Methods

M. CenterNet [2019, Zhou]

➢ Structure: Whole



III. Other Methods

M. CenterNet [2019, Zhou]

- Hourglass: 1540, 2016, Newell
- Resnet + Transpose: 187, 2018, Xiao
- DLA: 145, 2018(2019), Yu
- Modified DLA: [This Paper]

➢ Structure: Backbone

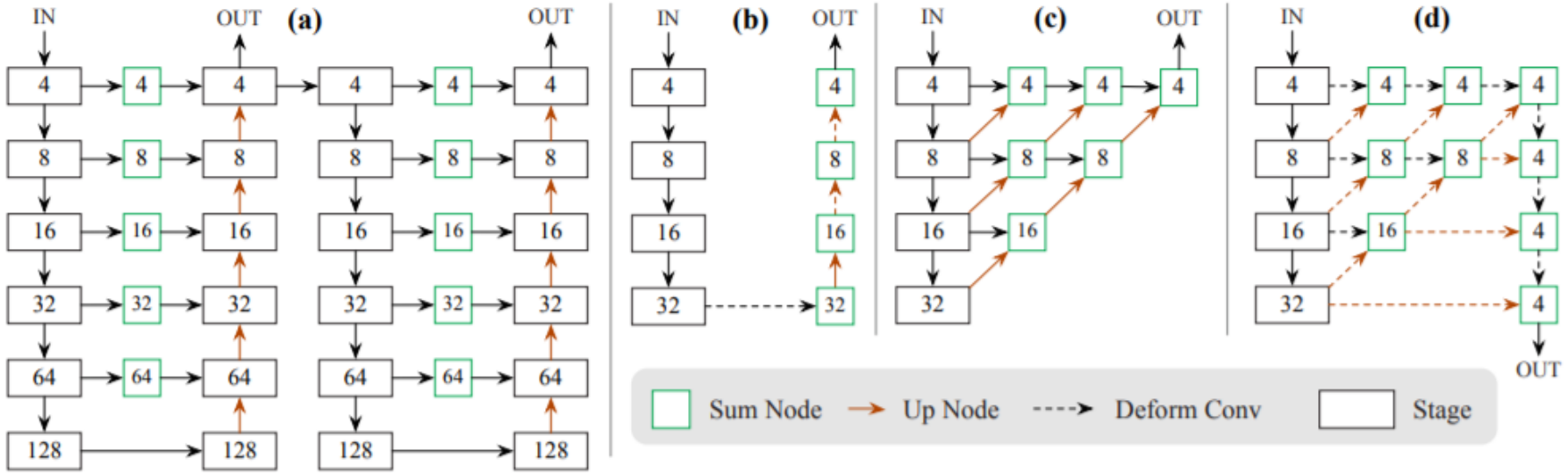
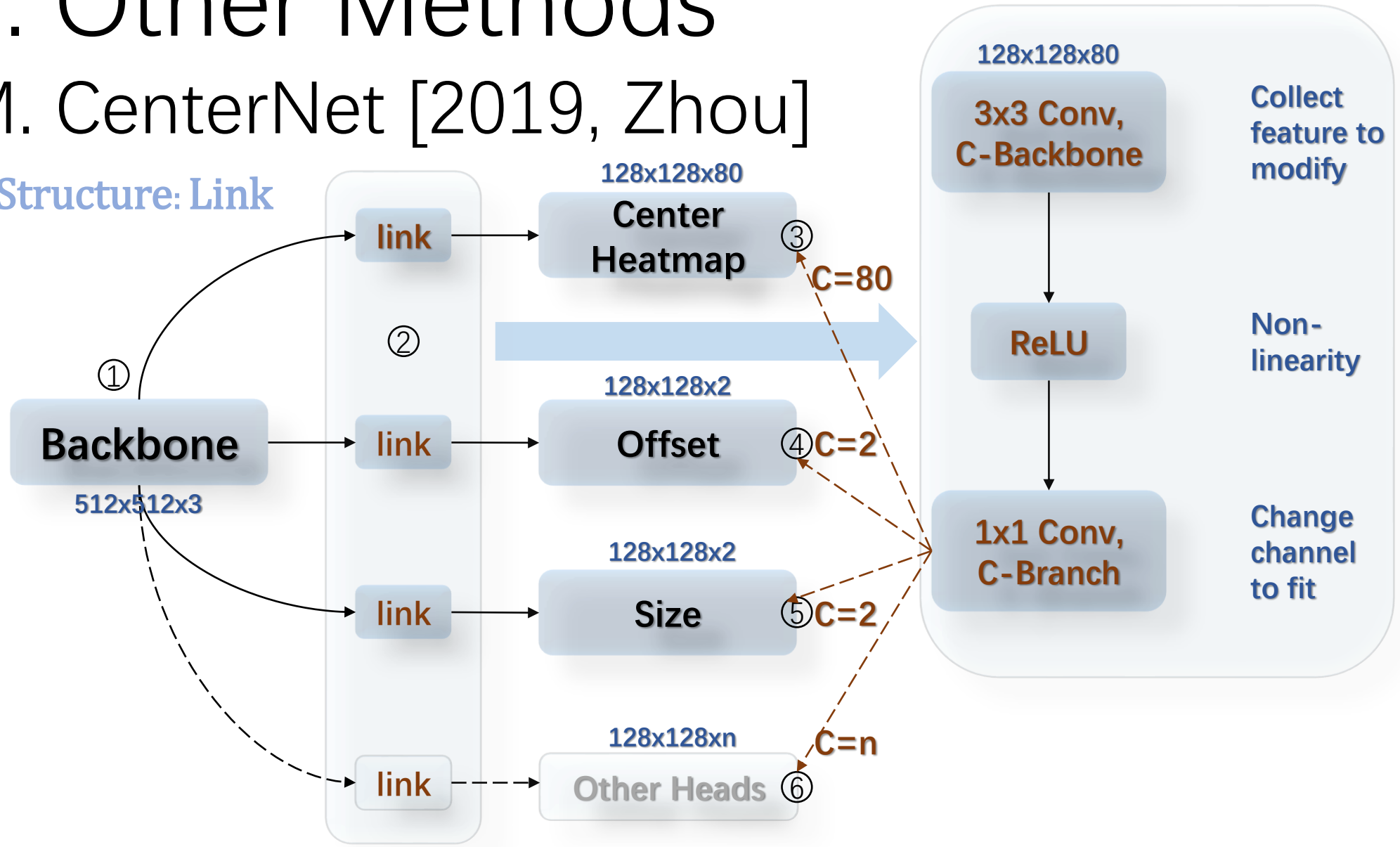


Figure 6: Model diagrams. The numbers in the boxes represent the stride to the image. (a): Hourglass Network [30]. We use it as is in CornerNet [30]. (b): ResNet with transpose convolutions [55]. We add one 3×3 deformable convolutional layer [63] before each up-sampling layer. Specifically, we first use deformable convolution to change the channels and then use transposed convolution to upsample the feature map (such two steps are shown separately in $32 \rightarrow 16$. We show these two steps together as a dashed arrow for $16 \rightarrow 8$ and $8 \rightarrow 4$). (c): The original DLA-34 [58] for semantic segmentation. (d): Our modified DLA-34. We add more skip connections from the bottom layers and upgrade every convolutional layer in upsampling stages to deformable convolutional layer.

III. Other Methods

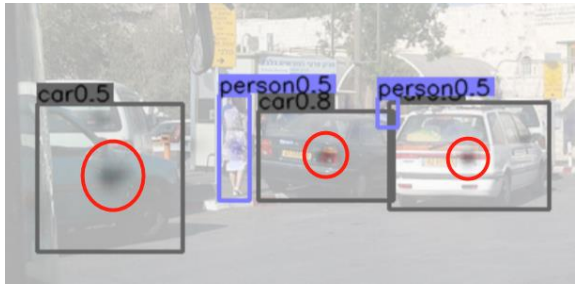
M. CenterNet [2019, Zhou]

➢ Structure: Link



III. Other Methods

M. CenterNet [2019, Zhou]



- **Depict objects using their center points**
- **Describe center point by heatmap**

$$Y_{xyc} = exp\left(-\frac{(x-\widetilde{p_x})^2+(y-\widetilde{p_y})^2}{2\sigma_p^2}\right) \in [0,1]^{\frac{W}{R}\times\frac{H}{R}\times C},$$

$$\tilde{p} = \left\lfloor \frac{p=original\ center\ coord}{R=4} \right\rfloor,$$

σ_p is determined by object size [CornerNet] [code]

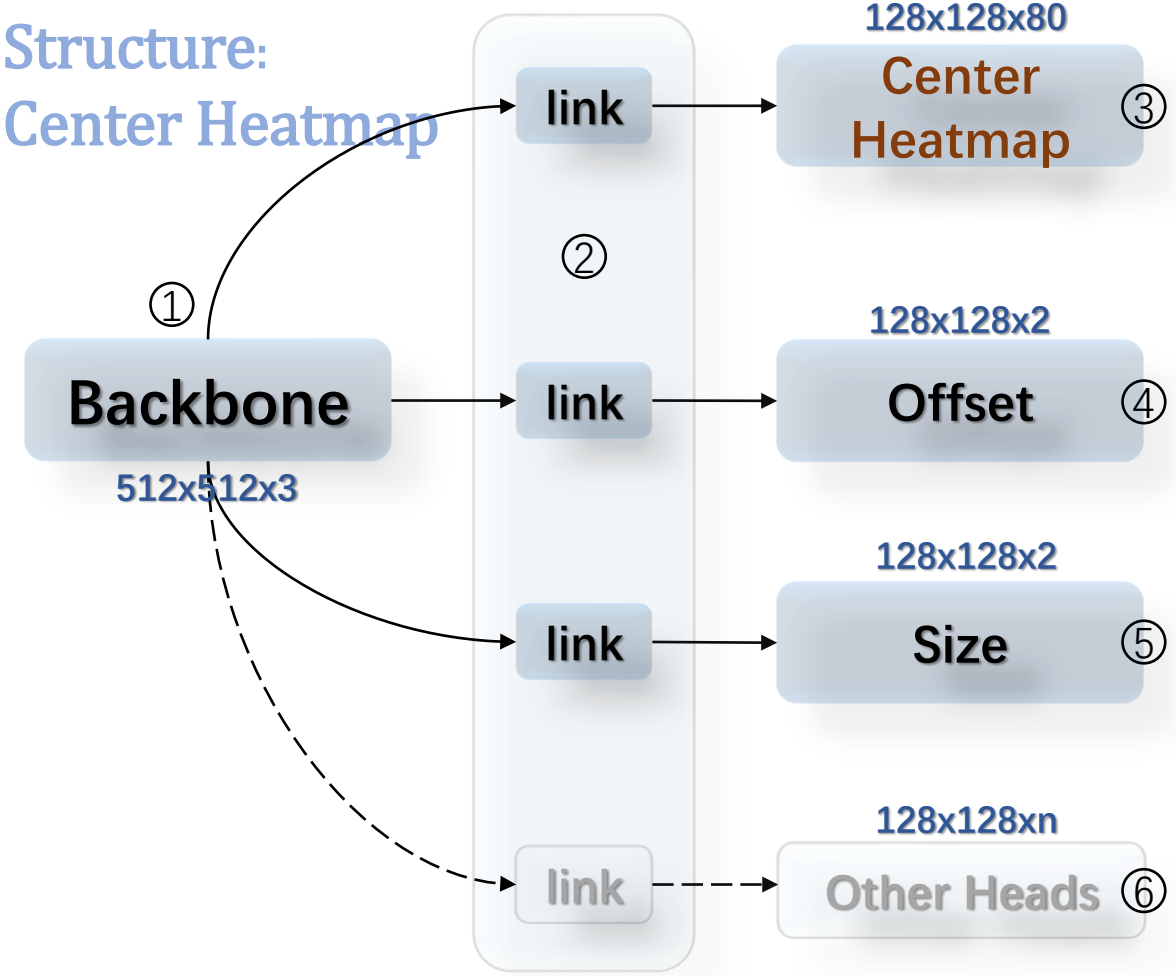
If overlap, take element-wise maximum

- **One class one channel**
- **Regress focal loss: $\alpha = 2, \beta = 4$**

$$L_k = \frac{-1}{N}\sum_{xyc}\begin{cases}\left(1-\hat{Y}_{xyc}\right)^{\alpha}\log\left(\hat{Y}_{xyc}\right), Y_{xyc}=1\\ \left(1-Y_{xyc}\right)^{\beta}\left(\hat{Y}_{xyc}\right)^{\alpha}\log\left(1-\hat{Y}_{xyc}\right), Y_{xyc}\neq 1\end{cases}$$

- **Reference: +3x3 max pool → remove NMS**

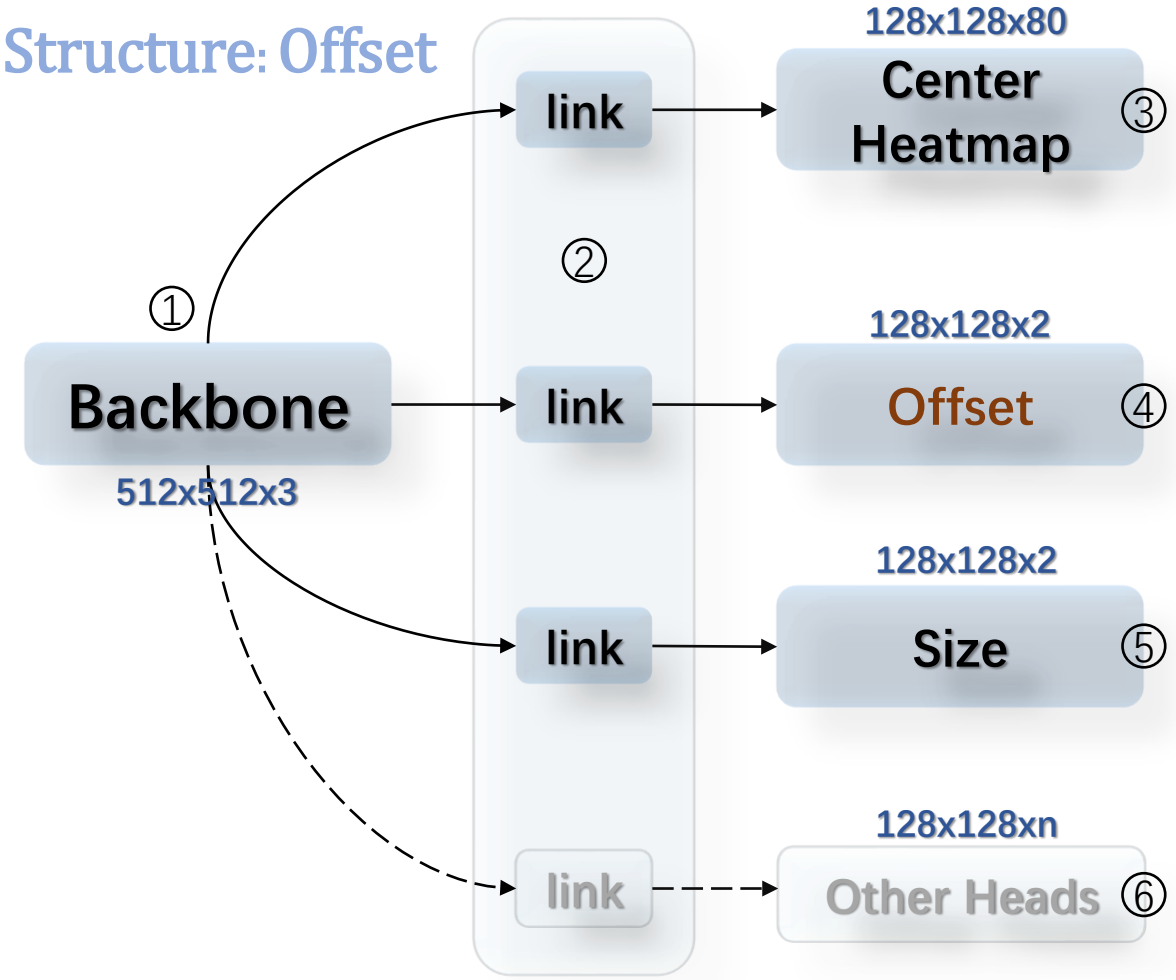
```
def _nms(heat, kernel=3):
    pad = (kernel - 1) // 2

    hmax = nn.functional.max_pool2d(
        heat, (kernel, kernel), stride=1, padding=pad)
    keep = (hmax == heat).float()
    return heat * keep
```



III. Other Methods

M. CenterNet [2019, Zhou]

➢ Structure: Offset

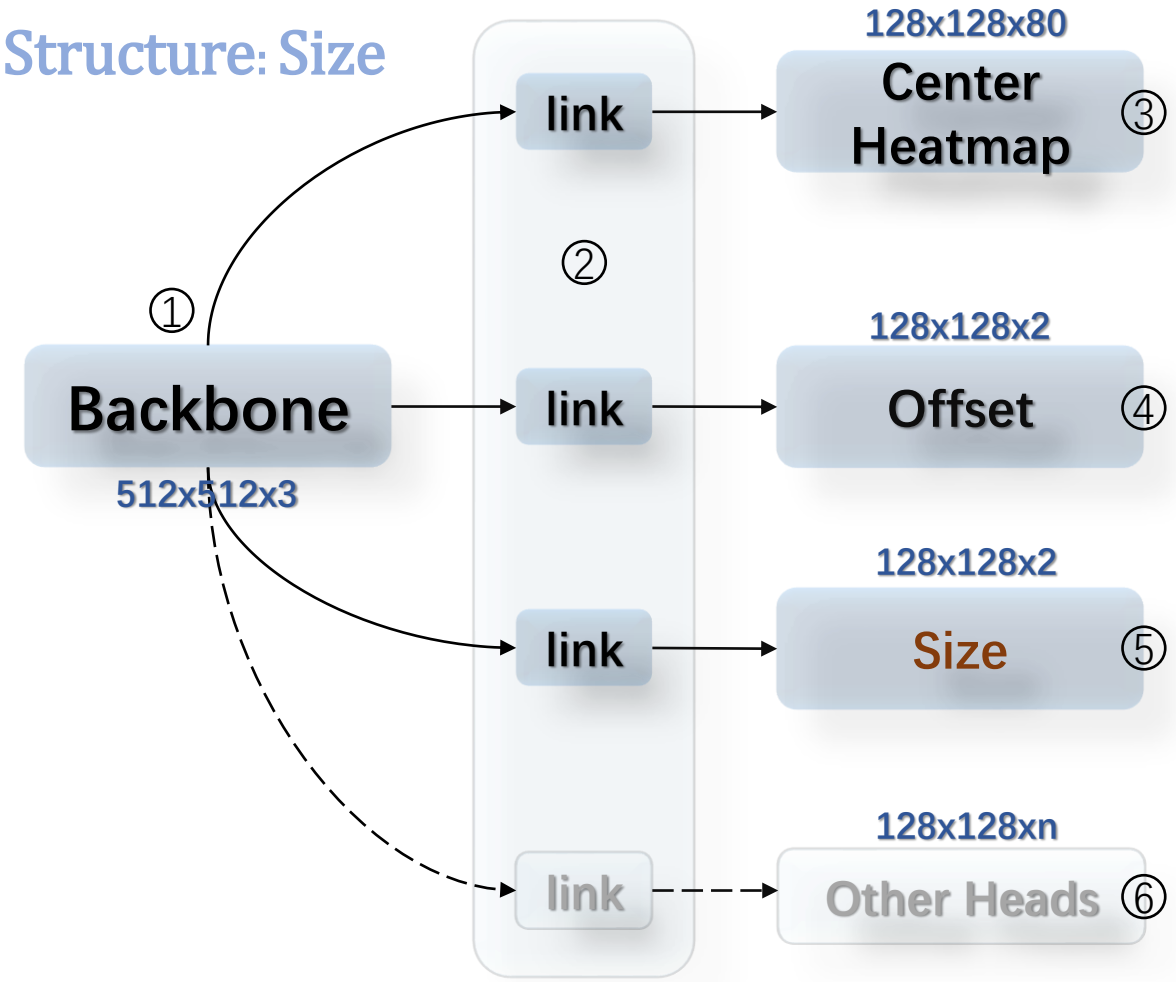


- **Target: Compensate discretization (512→128, /4)**
- **Predict 2-D offset:**
  $\hat{O}_{\hat{x}_i,\hat{y}_i} = (\delta\hat{x}_i, \delta\hat{y}_i)$
- **Regress L1 loss**
  $L_{off} = \frac{1}{N}\sum_p \left|\hat{O} - (\frac{p}{R} - \tilde{p})\right|$

III. Other Methods

M. CenterNet [2019, Zhou]

➢ Structure: Size



- **Predict BBox width & height**
- **Regress L1 loss**

$L_{size} = \frac{1}{N}\sum_{k=1}^{N}\left|\hat{S}_{p_k} - s_k\right|,$

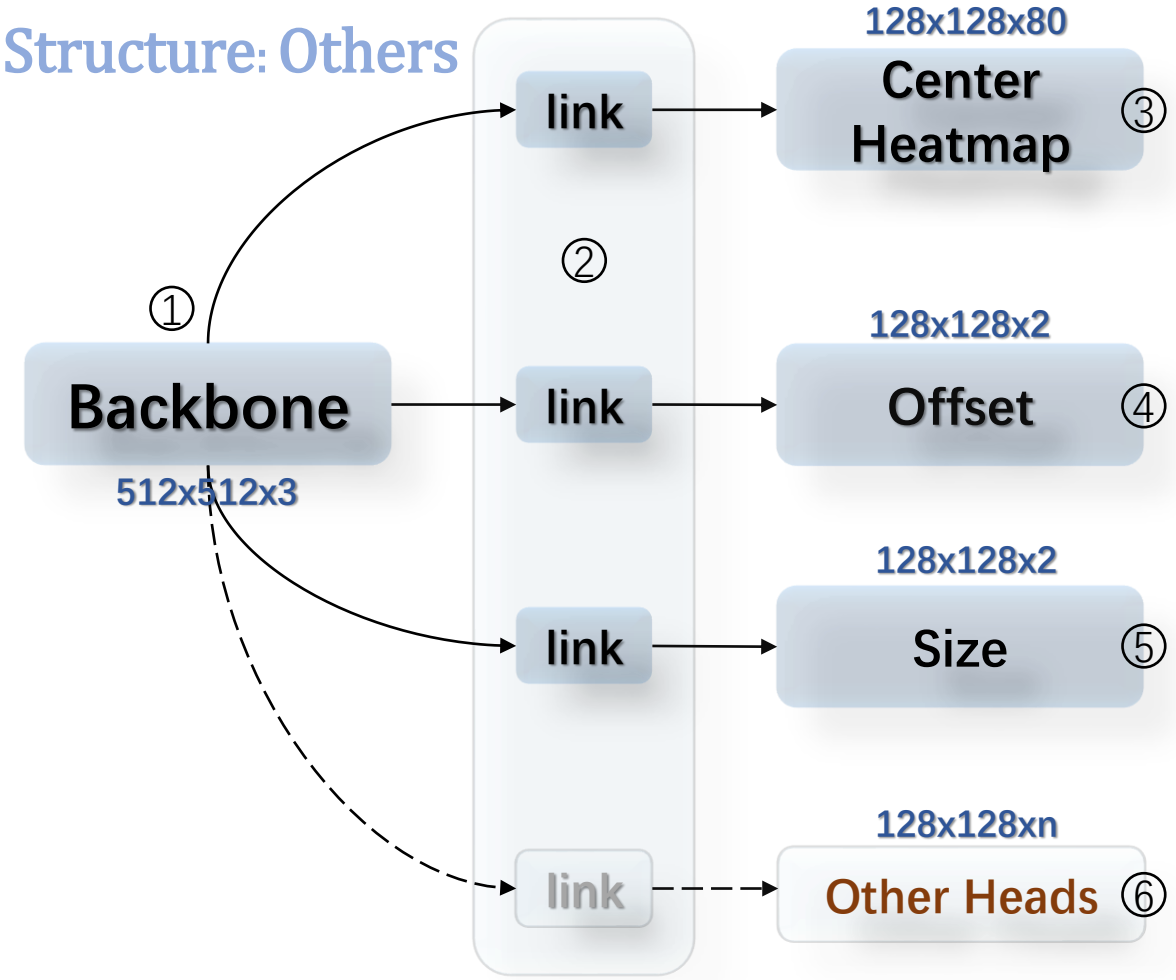
$s_k = \left(x_2^{(k)} - x_1^{(k)}, y_2^{(k)} - y_1^{(k)}\right),$

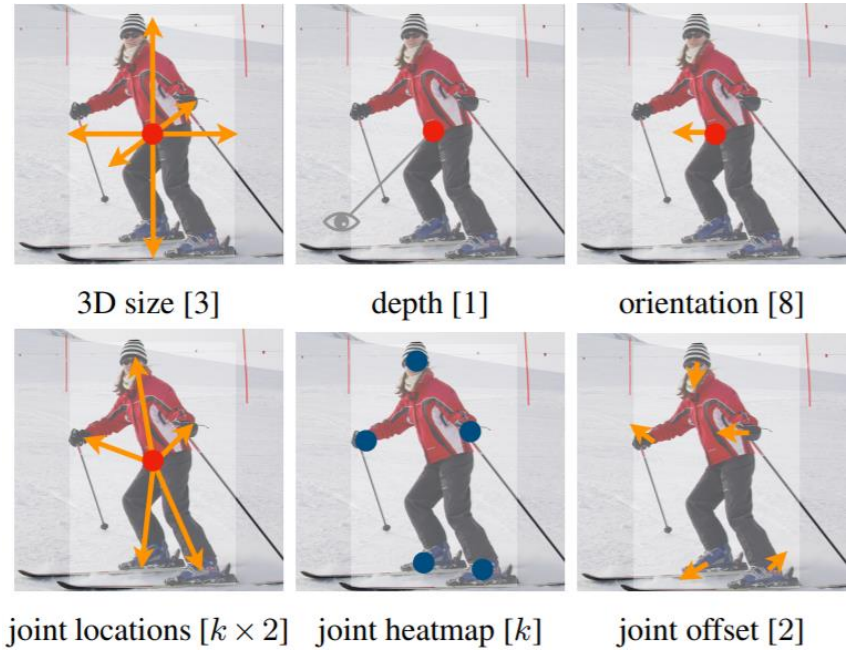
$k = id\ in\ class$

III. Other Methods

M. CenterNet [2019, Zhou]

➢ Structure: Others





III. Other Methods

L. CenterNet [2019, Zhou]

➢ Train:

128x128x80 → **Center Heatmap**

$$L_k = \frac{-1}{N}\sum_{xyc}\begin{cases}(1-\hat{Y}_{xyc})^{\alpha}\log(\hat{Y}_{xyc}), Y_{xyc}=1\\(1-Y_{xyc})^{\beta}(\hat{Y}_{xyc})^{\alpha}\log(1-\hat{Y}_{xyc}), Y_{xyc}\neq 1\end{cases}$$

128x128x2 → **Offset**

$$L_{off} = \frac{1}{N}\sum_p \left|\hat{O} - (\frac{p}{R} - \tilde{p})\right|$$

128x128x2 → **Size**

$$L_{size} = \frac{1}{N}\sum_{k=1}^{N}\left|\hat{S}_{p_k} - s_k\right|$$

$$L_{det} = L_k + \lambda_{size}L_{size} + \lambda_{off}L_{off}, (\lambda_{size} = 0.1, \lambda_{off} = 1)$$

- Use Adam, 10x learning rate dropped during training twice.
- Random flip, scaling(0.6~1.3), cropping, color jittering as augmentation
- Down sampling network structures are pretrained using ImageNet
- Different net has different LR initialization

III. Other Methods

M. CenterNet [2019, Zhou]

- 3 test augmentations:
  x, flip, flip + multi-scale
  (0.5, 0.75, 1, 1.25, 1.5)
- Do NMS when multi-scale

➢ Test:

	AP			AP_{50}			AP_{75}			Time (ms)			FPS		
	N.A.	F	MS	N.A.	F	MS	N.A.	F	MS	N.A.	F	MS	N.A.	F	MS
Hourglass-104	**40.3**	**42.2**	**45.1**	**59.1**	**61.1**	**63.5**	**44.0**	**46.0**	**49.3**	71	129	672	14	7.8	1.4
DLA-34	37.4	39.2	41.7	55.1	57.0	60.1	40.8	42.7	44.9	19	36	248	52	28	4
ResNet-101	34.6	36.2	39.3	53.0	54.8	58.5	36.9	38.7	42.0	22	40	259	45	25	4
ResNet-18	28.1	30.0	33.2	44.9	47.5	51.5	29.6	31.6	35.1	**7**	**14**	**81**	**142**	**71**	**12**

Resolution	AP	AP_{50}	AP_{75}	Time
Original	**36.3**	54.0	**39.6**	19
512	36.2	**54.3**	38.7	16
384	33.2	50.5	35.0	**11**

(a) Testing resolution: Lager resolutions perform better but run slower.

λ_{size}	AP	AP_{50}	AP_{75}
0.2	33.5	49.9	36.2
0.1	**36.3**	54.0	**39.6**
0.02	35.4	**54.6**	37.9

(b) Size regression weight. $\lambda_{size} \leq 0.1$ yields good results.

Loss	AP	AP_{50}	AP_{75}
l1	**36.3**	**54.0**	**39.6**
smooth l1	33.9	50.9	36.8

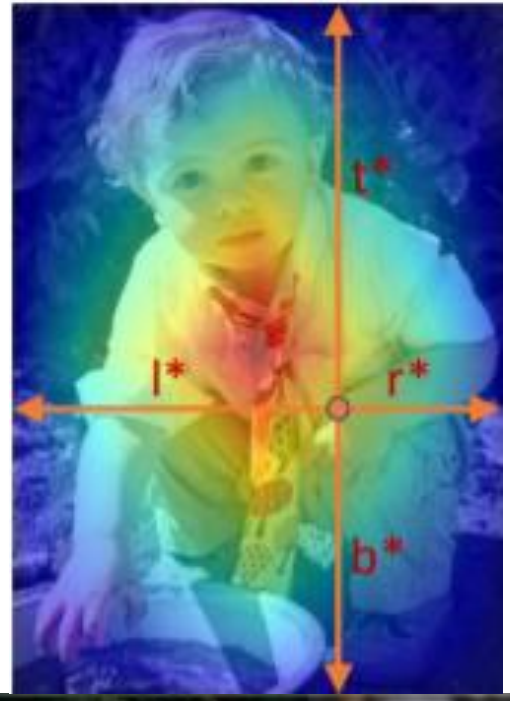
(c) Regression loss. L1 loss works better than Smooth L1.

Epoch	AP	AP_{50}	AP_{75}
140	36.3	54.0	39.6
230	**37.4**	**55.1**	**40.8**

(d) Training schedule. Longer performs better.

III. Other Methods

N. FCOS [2019, Tian]



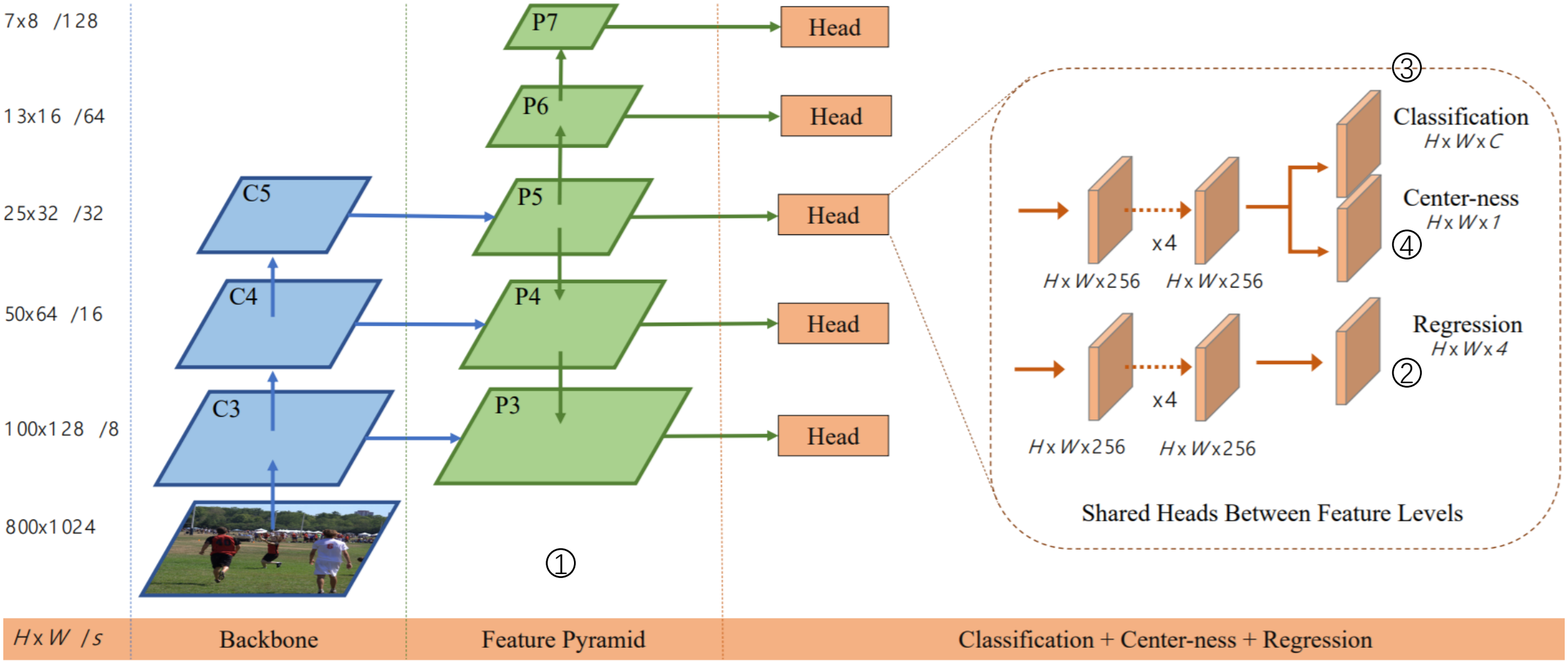
➢ Features:

- Detect objects by pixels
  [detect all points with an object]
- FPN as backbone
- Detect by scale
- Fully convolution one-stage detection
- Need NMS for post-processing

III. Other Methods

N. FCOS [2019, Tian]

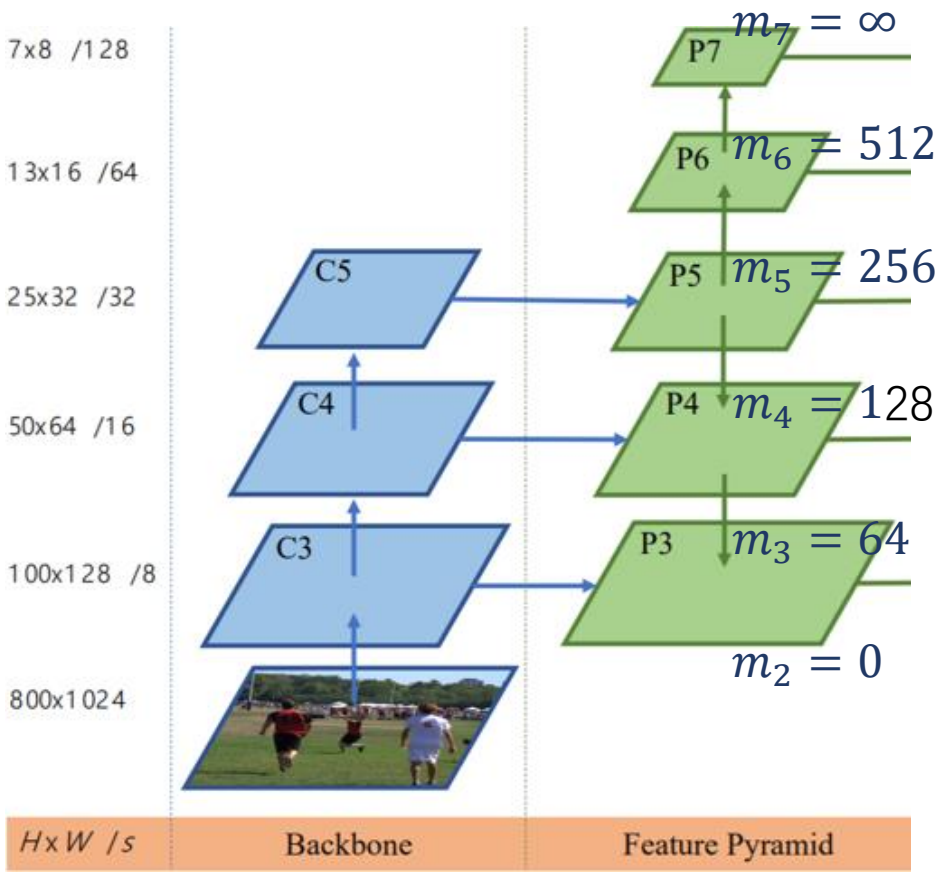
➢ Structure: Whole

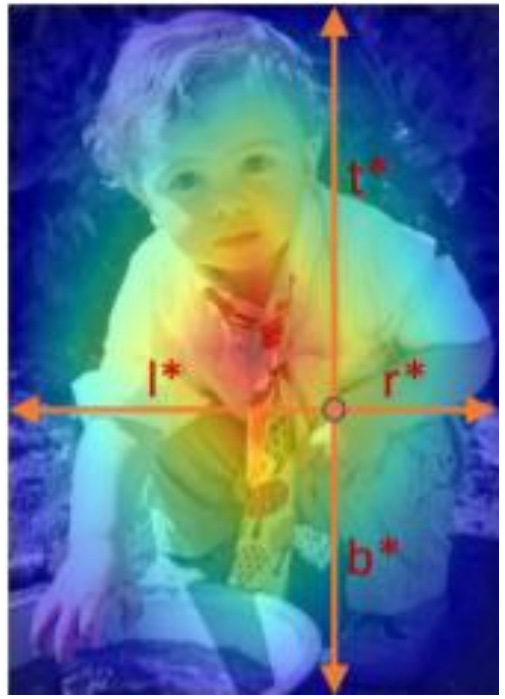


III. Other Methods

N. FCOS [2019, Tian]

➢ Structure: FPN ①



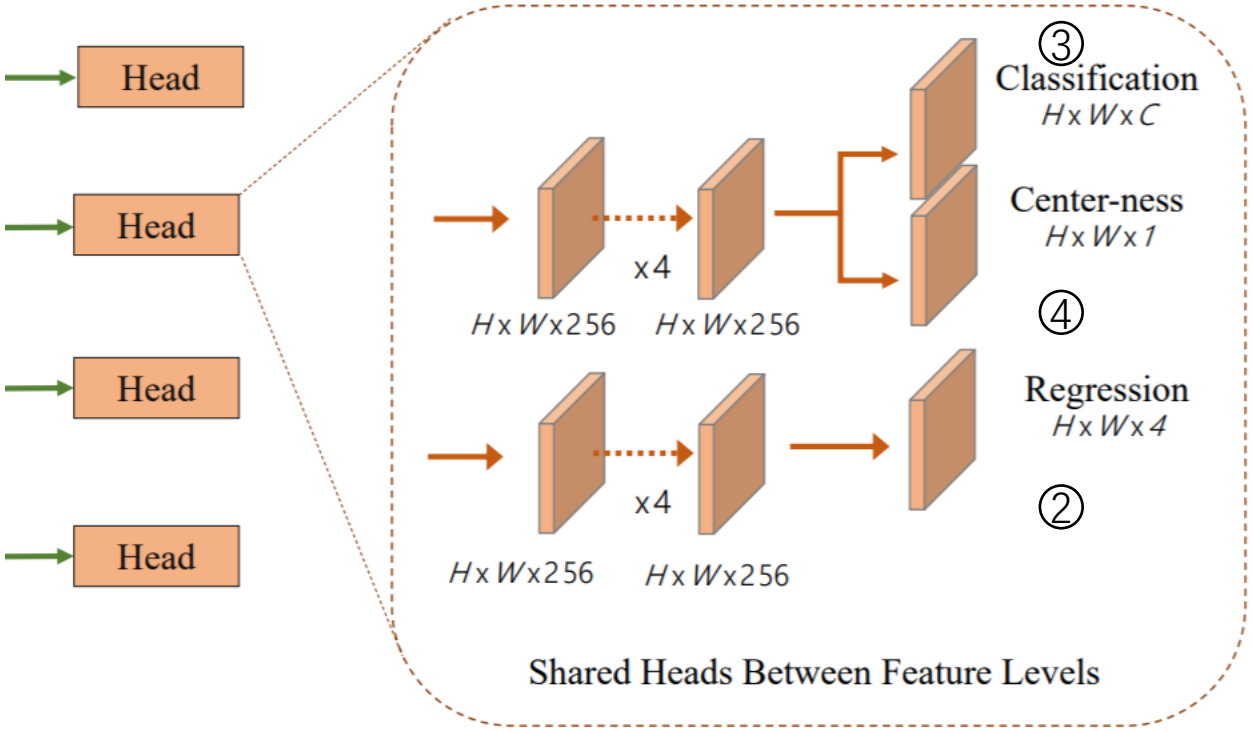


- **FPN: [Lin, 2016]**
- Pi: $m_{i-1} < max(l^*, r^*, t^* b^*) < m_i$
- $(l^*, r^*, t^* b^*)$, **denoted by** t^*, **is calculated on feature map for each feature pixel which can be mapped inside the ground truth bbox by using method:** $(\lfloor \frac{s}{2} + xs \rfloor, \lfloor \frac{s}{2} + ys \rfloor)$, s **is the stride**
- *: **ground truth**

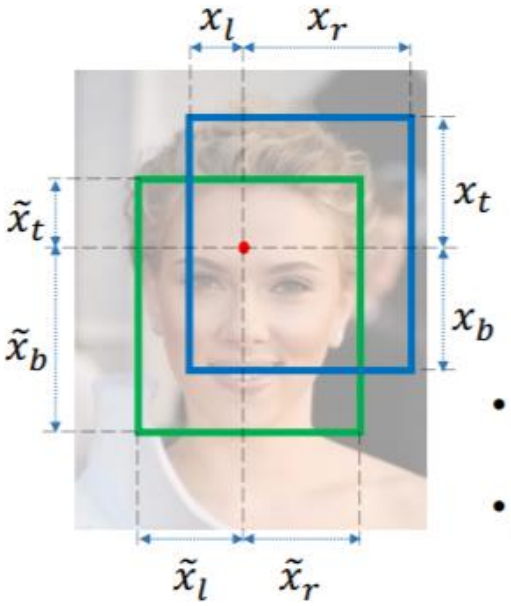
III. Other Methods

N. FCOS [2019, Tian]

➢ Structure: Reg ②



- $IoU\ loss$: **UnitBox [Yu, CVPR 2016]**
- $L_{reg} = IoU\ loss(t_{x,y}, t^*_{x,y})$



Ground truth: $\tilde{x} = (\tilde{x}_t, \tilde{x}_b, \tilde{x}_l, \tilde{x}_r)$

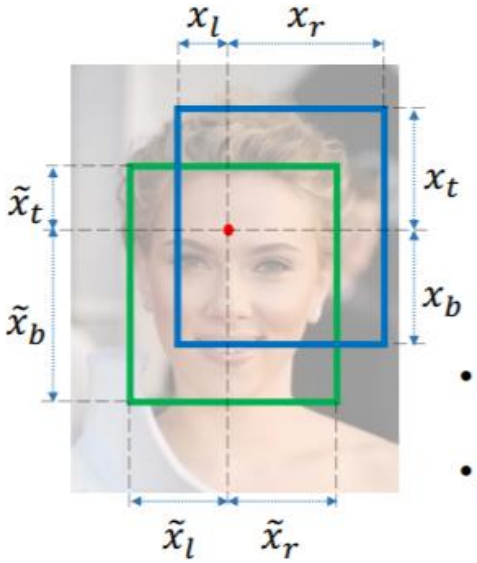
Prediction: $x = (x_t, x_b, x_l, x_r)$

- $\ell_2\ \ loss = ||\square - \square||_2^2$
- $IoU\ loss = -\ln \frac{Intersection(\square, \square)}{Union(\square, \square)}$

III. Other Methods

N. FCOS [2019, Tian]

➢ Structure: Reg ② — IoU Loss Forward



Ground truth: $\tilde{x} = (\tilde{x}_t, \tilde{x}_b, \tilde{x}_l, \tilde{x}_r)$

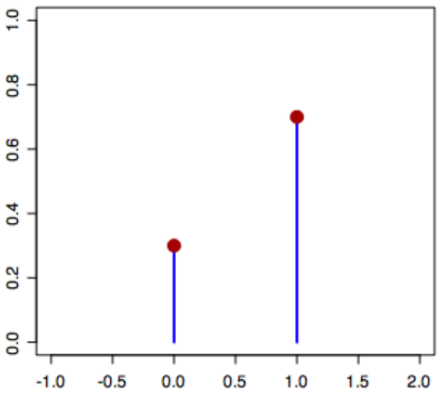
Prediction: $x = (x_t, x_b, x_l, x_r)$

- $\ell_2 \ \ loss = ||\square - \square||_2^2$
- $IoU\ loss = -\ln \frac{Intersection(\square, \square)}{Union(\square, \square)}$

Algorithm 1: *IoU* loss Forward

Input: $\tilde{\boldsymbol{x}}$ as bounding box ground truth
Input: $\boldsymbol{x}$ as bounding box prediction
Output: $\mathcal{L}$ as localization error
for *each pixel* (i, j) **do**
  if $\tilde{\boldsymbol{x}} \neq \mathbf{0}$ **then**
    $X = (x_t + x_b) * (x_l + x_r)$
    $\tilde{X} = (\tilde{x}_t + \tilde{x}_b) * (\tilde{x}_l + \tilde{x}_r)$
    $I_h = min(x_t, \tilde{x}_t) + min(x_b, \tilde{x}_b)$
    $I_w = min(x_l, \tilde{x}_l) + min(x_r, \tilde{x}_r)$
    $I = I_h * I_w$
    $U = X + \tilde{X} - I$
    $IoU = \frac{I}{U}$
    $\mathcal{L} = -ln(IoU)$
  else
    $\mathcal{L} = 0$
  end
end

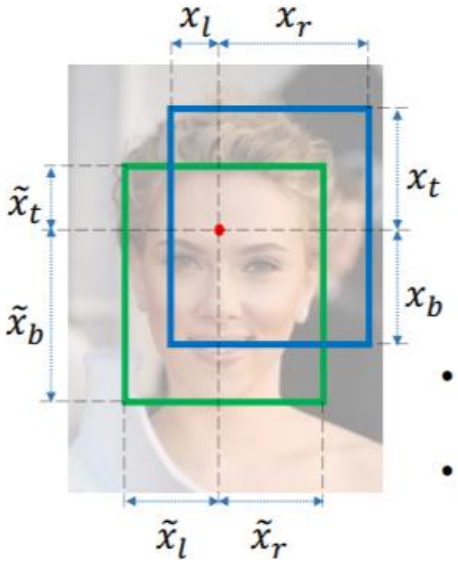
- $\tilde{x} \neq 0$, **pixel(i,j) falls inside a valid bbox**
- $IoU \in [0,1]$
- $L = -ln(IoU) = -pln(IoU) - (1-p)ln(1-IoU)$
- **Let** $p(IoU = 1) = 1$ **when** IoU **under Bernoulli distribution**



III. Other Methods

N. FCOS [2019, Tian]

➢ Structure: Reg ② — IoU Loss Backward



Ground truth: $\tilde{x} = (\tilde{x}_t, \tilde{x}_b, \tilde{x}_l, \tilde{x}_r)$

Prediction: $x = (x_t, x_b, x_l, x_r)$

- $\ell_2 \ \ loss = ||\square - \square||_2^2$
- $IoU\ loss = -\ln \frac{Intersection(\square, \square)}{Union(\square, \square)}$

Algorithm 1: *IoU* loss Forward

Input: $\tilde{\boldsymbol{x}}$ as bounding box ground truth
Input: $\boldsymbol{x}$ as bounding box prediction
Output: $\mathcal{L}$ as localization error
for *each pixel* (i, j) **do**
  if $\tilde{\boldsymbol{x}} \neq \mathbf{0}$ **then**
    $X = (x_t + x_b) * (x_l + x_r)$
    $\tilde{X} = (\tilde{x}_t + \tilde{x}_b) * (\tilde{x}_l + \tilde{x}_r)$
    $I_h = min(x_t, \tilde{x}_t) + min(x_b, \tilde{x}_b)$
    $I_w = min(x_l, \tilde{x}_l) + min(x_r, \tilde{x}_r)$
    $I = I_h * I_w$
    $U = X + \tilde{X} - I$
    $IoU = \frac{I}{U}$
    $\mathcal{L} = -ln(IoU)$
  else
    $\mathcal{L} = 0$
  end
end

$$\frac{\partial \mathcal{L}}{\partial x} = \frac{I(\nabla_x X - \nabla_x I) - U\nabla_x I}{U^2 IoU} = \frac{1}{U}\nabla_x X - \frac{U+I}{UI}\nabla_x I.$$

How to make your own loss function?

$$\frac{\partial X}{\partial x_t (\text{or } \partial x_b)} = x_l + x_r$$

$$\frac{\partial X}{\partial x_l (\text{or } \partial x_r)} = x_t + x_b$$

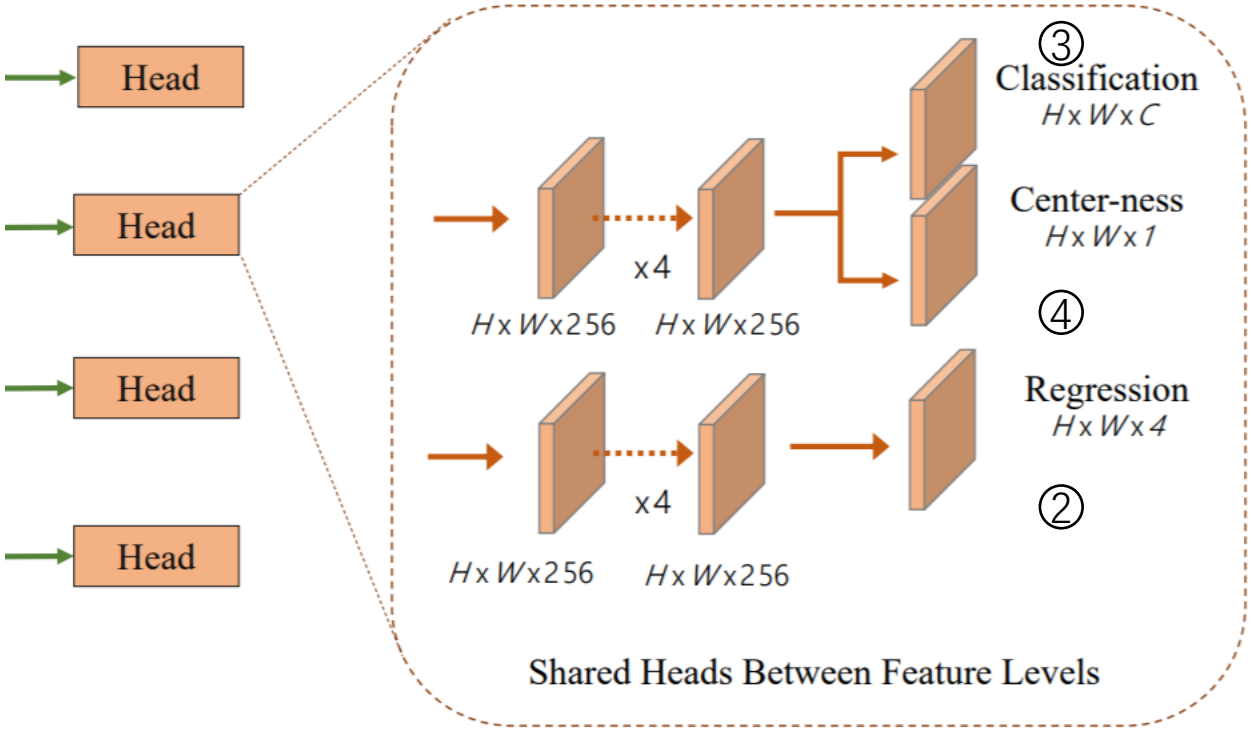
$$\frac{\partial I}{\partial x_t (\text{or } \partial x_b)} = \begin{cases} I_w, & \text{if } x_t < \tilde{x}_t (\text{or } x_b < \tilde{x}_b) \\ 0, & \text{otherwise,} \end{cases}$$

$$\frac{\partial I}{\partial x_l (\text{or } \partial x_r)} = \begin{cases} I_h, & \text{if } x_l < \tilde{x}_l (\text{or } x_r < \tilde{x}_r) \\ 0, & \text{otherwise.} \end{cases}$$

III. Other Methods

N. FCOS [2019, Tian]

➢ Structure: Cls ③

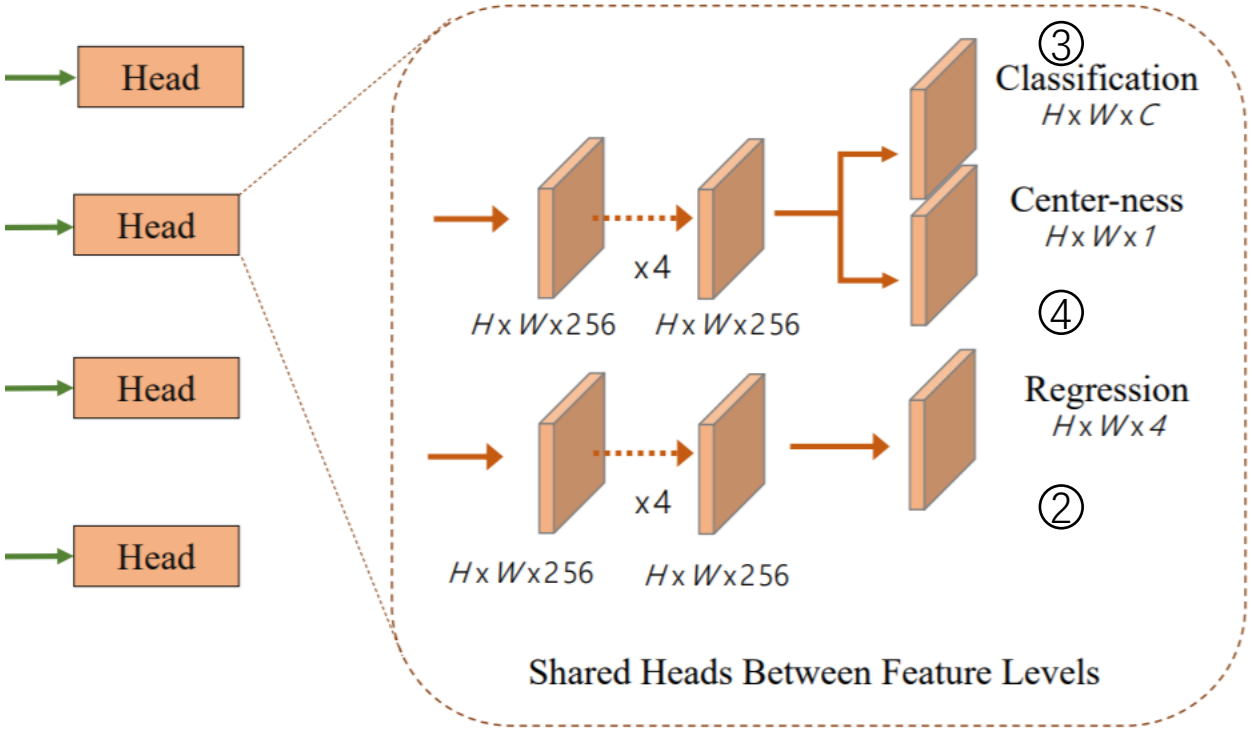


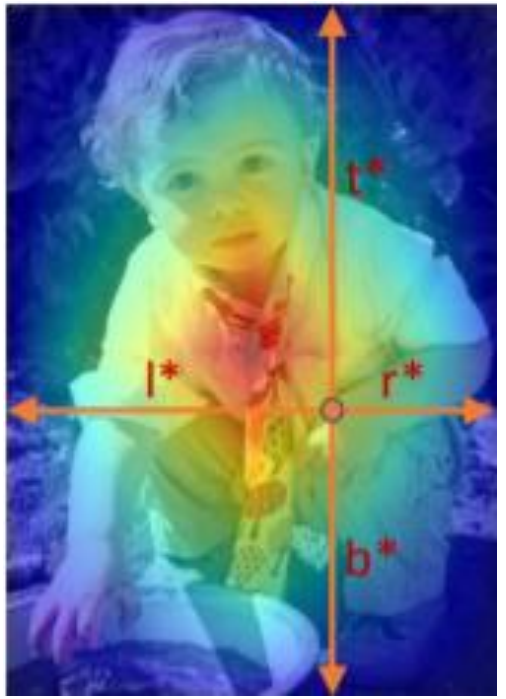
- *Cls loss*: **Focal Loss**
- $L_{cls} = -\alpha_t(1-p_t)^{\gamma}\log(p_t)$
  $[\gamma = 2, \alpha = 0.25]$

III. Other Methods

N. FCOS [2019, Tian]

➢ Structure: Center-ness ④





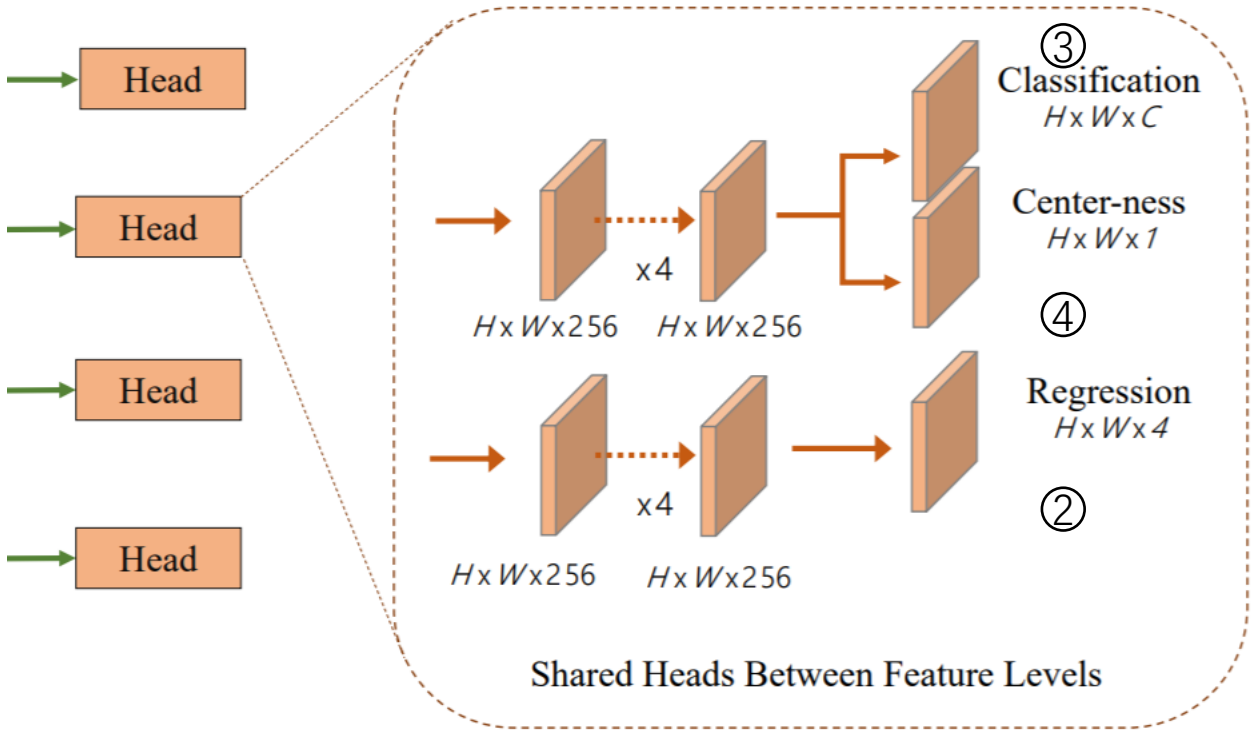
- **Target: fix low-quality border detection**
- **Set a score to each pixel**
- $centerness^* = \sqrt{\frac{\min(l^*,r^*)}{\max(l^*,r^*)} \times \frac{\min(t^*,b^*)}{\max(t^*,b^*)}}$

III. Other Methods

N. FCOS [2019, Tian]

➢ Train:

$$L_{p,t} = \frac{1}{N_{pos}} \sum_{x,y} L_{cls}(p_{x,y}, c^*_{x,y}) + \frac{\lambda}{N_{pos}} \mathbb{I}_{\{c^*_{x,y}>0\}} \left(\sum_{x,y} L_{reg}(t_{x,y}, t^*_{x,y}) + \sum_{x,y} L_{cent}(ct_{x,y}, ct^*_{x,y}) \right)$$



$$L_{cls} = -\alpha_t (1 - p_t)^{\gamma} \log(p_t)$$

$$L_{cent} = BCE(ct^*, ct)$$

$$L_{reg} = IoU\ loss(t_{x,y}, t^*_{x,y})$$