NOH-NMS: Improving Pedestrian Detection by Nearby Objects Hallucination

Tencent Youtu Lab(ACM MM 2020)

分享者：孙路

2024 年 1 月 3 日

Introduction

- Non-maximum Suppression (NMS) 是一个经典的后处理算法;

Introduction

- Non-maximum Suppression (NMS) 是一个经典的后处理算法；
- Greedy-NMS，N 个 box，处理复杂度 $\mathcal{O}(N^2)$，劣势在于对阈值敏感；

Introduction

- Non-maximum Suppression (NMS) 是一个经典的后处理算法；
- Greedy-NMS，N 个 box，处理复杂度 $\mathcal{O}(N^2)$，劣势在于对阈值敏感；
- NOH-NMS 利用密度以及参数信息修正阈值，从而达到更高的准确性；

Introduction

- Non-maximum Suppression (NMS) 是一个经典的后处理算法；
- Greedy-NMS，N 个 box，处理复杂度 $\mathcal{O}(N^2)$，劣势在于对阈值敏感；
- NOH-NMS 利用密度以及参数信息修正阈值，从而达到更高的准确性；
- CityPersons 以及 CrowdHuman 数据集上达到了 89%AP，92.9%Recall。

Greedy-NMS

最初的起点，基本原理是贪心，时间复杂度为 $\mathcal{O}(N^2)$。

input : $\mathcal{B} = b_1, \cdots, b_N, \mathcal{S} = s_1, \cdots, s_N, N_t$

$\mathcal{B}$ is the list of initial detection boxes, $\mathcal{S}$ contains corresponding detection scores, N_t is the NMS threshold

begin

$\mathcal{F} \leftarrow \{\}$;

while $\mathcal{B} \neq \varnothing$ **do**

$m \leftarrow \arg\max \mathcal{S}$;

$\mathcal{M} \leftarrow b_m$;

$\mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{M}; \mathcal{B} \leftarrow \mathcal{B} - \mathcal{M}$;

for $b_i \in \mathcal{B}$ **do**

if $\mathrm{iou}(\mathcal{M}, b_i) \geq N_t$ **then**

$B \leftarrow B - b_i; S \leftarrow S - s_i$;

end

end

end

return $\mathcal{F}, \mathcal{S}$

end

Soft-NMS

- Greedy-NMS 若 iou 大于阈值，则直接删去该框，容易将置信度高的临近框删去；

Soft-NMS

- Greedy-NMS 若 iou 大于阈值，则直接删去该框，容易将置信度高的临近框删去；
- 剔除过程过程 Soft 一下；

Soft-NMS

- Greedy-NMS 若 iou 大于阈值，则直接删去该框，容易将置信度高的临近框删去；
- 剔除过程过程 Soft 一下；
- 若 iou 大于阈值，改为乘上一个惩罚因子。

Soft-NMS

- Greedy-NMS 若 iou 大于阈值，则直接删去该框，容易将置信度高的临近框删去；
- 剔除过程过程 Soft 一下；
- 若 iou 大于阈值，改为乘上一个惩罚因子。
- Soft-NMS – Improving Object Detection With One Line of Code(ICCV 2017)

Soft-NMS

- Greedy-NMS 若 iou 大于阈值，则直接删去该框，容易将置信度高的临近框删去；
- 剔除过程过程 Soft 一下；
- 若 iou 大于阈值，改为乘上一个惩罚因子。
- Soft-NMS – Improving Object Detection With One Line of Code(ICCV 2017)

Soft-NMS

- Greedy-NMS 若 iou 大于阈值，则直接删去该框，容易将置信度高的临近框删去；
- 剔除过程过程 Soft 一下；
- 若 iou 大于阈值，改为乘上一个惩罚因子。
- Soft-NMS – Improving Object Detection With One Line of Code(ICCV 2017)

Box_i 的惩罚因子 $f(\mathcal{M}, b_i)$

$$f(\mathcal{M}, b_i) = \begin{cases} 1, & \text{iou}(\mathcal{M}, b_i) < N_t \\ 1 - \text{iou}(\mathcal{M}, b_i) \ \textbf{or} \ \exp(\frac{-\text{iou}(\mathcal{M}, b_i)^2}{\sigma}), & \text{iou}(\mathcal{M}, b_i) \geq N_t \end{cases} \tag{1}$$

Soft-NMS

每次至少删去一个 Box，时间复杂度仍为 $\mathcal{O}(N^2)$，但是运行效率大打折扣。

Soft-NMS

每次至少删去一个 Box，时间复杂度仍为 $\mathcal{O}(N^2)$，但是运行效率大打折扣。

input : $\mathcal{B} = b_1, \cdots, b_N, \mathcal{S} = s_1, \cdots, s_N, N_t$
$\mathcal{B}$ is the list of initial detection boxes, $\mathcal{S}$ contains corresponding detection scores, N_t is the NMS threshold
begin
  $\mathcal{F} \leftarrow \{\}$;
  while $\mathcal{B} \neq \varnothing$ **do**
    $m \leftarrow \arg\max \mathcal{S}$;
    $\mathcal{M} \leftarrow b_m$;
    $\mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{M}; \mathcal{B} \leftarrow \mathcal{B} - \mathcal{M}$;
    for $b_i \in \mathcal{B}$ **do**
      $s_i \leftarrow s_i \cdot f(\mathcal{M}, b_i)$;
    end
  end
  return $\mathcal{F}, \mathcal{S}$
end

Adaptive-NMS

- 考虑到阈值在 NMS 算法中为定值，因此不能很好的处理不同的密度情况；

Adaptive-NMS

- 考虑到阈值在 NMS 算法中为定值，因此不能很好的处理不同的密度情况；
- CNN solves everything；

Adaptive-NMS

- 考虑到阈值在 NMS 算法中为定值，因此不能很好的处理不同的密度情况；
- CNN solves everything；
- 训练一个并行的网络去预测密度信息 $d_{\mathcal{M}}$，不同的框根据密度设置不同的阈值；

Adaptive-NMS

- 考虑到阈值在 NMS 算法中为定值，因此不能很好的处理不同的密度情况；
- CNN solves everything；
- 训练一个并行的网络去预测密度信息 $d_{\mathcal{M}}$，不同的框根据密度设置不同的阈值；
- Adaptive NMS: Refining Pedestrian Detection in a Crowd(CVPR 2019)

Adaptive-NMS

- 考虑到阈值在 NMS 算法中为定值，因此不能很好的处理不同的密度情况;
- CNN solves everything;
- 训练一个并行的网络去预测密度信息 $d_{\mathcal{M}}$，不同的框根据密度设置不同的阈值;
- Adaptive NMS: Refining Pedestrian Detection in a Crowd(CVPR 2019)

Adaptive-NMS

- 考虑到阈值在 NMS 算法中为定值，因此不能很好的处理不同的密度情况；
- CNN solves everything；
- 训练一个并行的网络去预测密度信息 $d_{\mathcal{M}}$，不同的框根据密度设置不同的阈值；
- Adaptive NMS: Refining Pedestrian Detection in a Crowd(CVPR 2019)

$\mathcal{M}$ 的阈值 $N_{\mathcal{M}}$

$$N_{\mathcal{M}} := \max(N_t, d_{\mathcal{M}}) \tag{2}$$

Adaptive-NMS

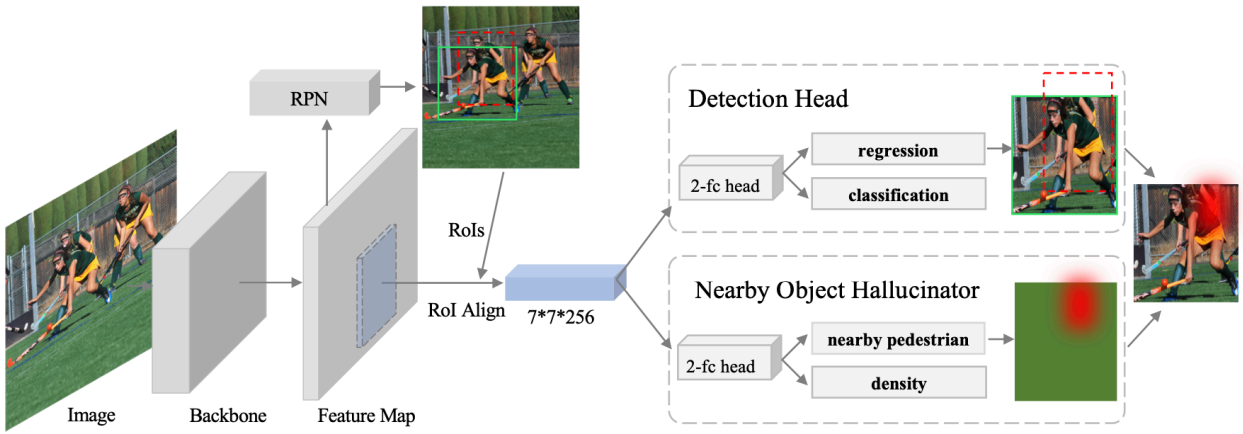

图 1: Adaptive-NMS 框架示意图

Adaptive-NMS

input : $\mathcal{B} = b_1, \cdots, b_N, \mathcal{S} = s_1, \cdots, s_N, \mathcal{D} = d_1, \cdots, d_N, N_t$

$\mathcal{B}$ is the list of initial detection boxes, $\mathcal{S}$ contains corresponding detection scores, $\mathcal{D}$ contains corresponding detection densities, N_t is the NMS threshold

begin

$\mathcal{F} \leftarrow \{\}$;

while $\mathcal{B} \neq \varnothing$ **do**

$m \leftarrow \arg\max \mathcal{S}$;

$\mathcal{M} \leftarrow b_m$;

$N_{\mathcal{M}} \leftarrow \max(N_t, d_m)$;

$\mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{M}; \mathcal{B} \leftarrow \mathcal{B} - \mathcal{M}$;

for $b_i \in \mathcal{B}$ **do**

if $\mathrm{iou}(\mathcal{M}, b_i) \geq N_{\mathcal{M}}$ **then**

$B \leftarrow B - b_i; S \leftarrow S - s_i$;

end

end

end

return $\mathcal{F}, \mathcal{S}$

end

Fast NMS

- NMS 太慢了，尤其是软处理系列的 NMS，由于几乎都是串行计算，算法常数很大；

Fast NMS

- NMS 太慢了，尤其是软处理系列的 NMS，由于几乎都是串行计算，算法常数很大；
- 牺牲一些精度换取更快的实现；

Fast NMS

- NMS 太慢了，尤其是软处理系列的 NMS，由于几乎都是串行计算，算法常数很大；
- 牺牲一些精度换取更快的实现；
- 会抑制更多的框，在密度大的情况下表现不好；

Fast NMS

- NMS 太慢了，尤其是软处理系列的 NMS，由于几乎都是串行计算，算法常数很大；
- 牺牲一些精度换取更快的实现；
- 会抑制更多的框，在密度大的情况下表现不好；
- 时间复杂度瓶颈在于上三角矩阵的计算，这一步还是 $\mathcal{O}(N^2)$；

Fast NMS

- NMS 太慢了，尤其是软处理系列的 NMS，由于几乎都是串行计算，算法常数很大；
- 牺牲一些精度换取更快的实现；
- 会抑制更多的框，在密度大的情况下表现不好；
- 时间复杂度瓶颈在于上三角矩阵的计算，这一步还是 $\mathcal{O}(N^2)$；
- YOLACT: Real-time Instance Segmentation(ICCV 2019)；

Fast NMS

- NMS 太慢了，尤其是软处理系列的 NMS，由于几乎都是串行计算，算法常数很大；
- 牺牲一些精度换取更快的实现；
- 会抑制更多的框，在密度大的情况下表现不好；
- 时间复杂度瓶颈在于上三角矩阵的计算，这一步还是 $\mathcal{O}(N^2)$；
- YOLACT: Real-time Instance Segmentation(ICCV 2019)；
- “In the Mask R-CNN benchmark suite , Fast NMS is 15.0 ms faster than their CUDA implementation of traditional NMS with a performance loss of only 0.3 mAP.”

Fast NMS

input : $\mathcal{B} = b_1, \cdots, b_N, \mathcal{S} = s_1, \cdots, s_N, N_t$

$\mathcal{B}$ is the list of initial detection boxes, $\mathcal{S}$ contains corresponding detection scores, N_t is the NMS threshold

begin

$\mathcal{F} \leftarrow \{\}$;

$\texttt{Sort}(\mathcal{B})$ in descending order based on the key value $\mathcal{S}$;

$\mathrm{IoU}_{N \times N} \leftarrow \mathbf{O}$;

for $i := 1$ *to* N **do**

  $t \leftarrow 0$;

  for $j := 1$ *to* $i - 1$ **do**

    $t := \max(t, \mathrm{iou}(b_i, b_j))$;

  end

  if $t < N_t$ **then**

    $\mathcal{F} \leftarrow \mathcal{F} \cup b_i$

  end

end

return $\mathcal{F}$

end

Other work

- Matrix NMS(SOLOv2: Dynamic and Fast Instance Segmentation, NeurIPS'20);

Other work

- Matrix NMS(SOLOv2: Dynamic and Fast Instance Segmentation, NeurIPS'20);
- Cluster NMS(Enhancing Geometric Factors in Model Learning and Inference for Object Detection and Instance Segmentation, TCYB)

NOH-NMS Methodology

- 结合 Adaptive-NMS 和 Soft NMS 的优势;

NOH-NMS Methodology

- 结合 Adaptive-NMS 和 Soft NMS 的优势；
- 密度大的时候不选用脉冲函数，而是乘以一个高斯分布。

NOH-NMS Methodology

- 结合 Adaptive-NMS 和 Soft NMS 的优势;
- 密度大的时候不选用脉冲函数，而是乘以一个高斯分布。

NOH-NMS Methodology

- 结合 Adaptive-NMS 和 Soft NMS 的优势；
- 密度大的时候不选用脉冲函数，而是乘以一个高斯分布。

NOH 惩罚因子 $f(\mathcal{M}, b_i, d_{\mathcal{M}}, p_{\mathcal{M}})$

$$f(\mathcal{M}, b_i, d_{\mathcal{M}}, p_{\mathcal{M}}) = \begin{cases} P_{P\mathcal{M}}(\mathcal{M}, b_i), & d_{\mathcal{M}} \geq d_t \\ 0, & d_{\mathcal{M}} < d_t \end{cases} \tag{3}$$

NOH-NMS Methodology

- 结合 Adaptive-NMS 和 Soft NMS 的优势；
- 密度大的时候不选用脉冲函数，而是乘以一个高斯分布。

NOH 惩罚因子 $f(\mathcal{M}, b_i, d_{\mathcal{M}}, p_{\mathcal{M}})$

$$f(\mathcal{M}, b_i, d_{\mathcal{M}}, p_{\mathcal{M}}) = \begin{cases} P_{P\mathcal{M}}(\mathcal{M}, b_i), & d_{\mathcal{M}} \geq d_t \\ 0, & d_{\mathcal{M}} < d_t \end{cases} \tag{3}$$

其中 d_t 是一个超参数，文中设置成 0.3。

Nearby Object Hallucinator

- Nearby Object Hallucinators 也是一个并行的计算密度的 CNN 网络;

Nearby Object Hallucinator

- Nearby Object Hallucinators 也是一个并行的计算密度的 CNN 网络;
- 同时它会预测出一些 hallucination;

Nearby Object Hallucinator

- Nearby Object Hallucinators 也是一个并行的计算密度的 CNN 网络;
- 同时它会预测出一些 hallucination;
- 考虑到 $\mathcal{M}$ 时，只考虑重叠最多的幻觉框，参数记为 $\mu_{\mathcal{M}}$。

Nearby Object Hallucinator

- Nearby Object Hallucinators 也是一个并行的计算密度的 CNN 网络;
- 同时它会预测出一些 hallucination;
- 考虑到 $\mathcal{M}$ 时，只考虑重叠最多的幻觉框，参数记为 $\mu_{\mathcal{M}}$。

Nearby Object Hallucinator

- Nearby Object Hallucinators 也是一个并行的计算密度的 CNN 网络;
- 同时它会预测出一些 hallucination;
- 考虑到 $\mathcal{M}$ 时，只考虑重叠最多的幻觉框，参数记为 $\mu_{\mathcal{M}}$。

分布参数 P

$$P_{\mu_{\mathcal{M}}}(\mathcal{M}, b_i) = \exp(-\|b_{i|\mathcal{M}} - \mu_{\mathcal{M}}\|^2 / 2\sigma^2) \tag{4}$$

Nearby Object Hallucinator

- Nearby Object Hallucinators 也是一个并行的计算密度的 CNN 网络;
- 同时它会预测出一些 hallucination;
- 考虑到 $\mathcal{M}$ 时，只考虑重叠最多的幻觉框，参数记为 $\mu_{\mathcal{M}}$。

分布参数 P

$$P_{\mu_{\mathcal{M}}}(\mathcal{M}, b_i) = \exp(-\|b_{i|\mathcal{M}} - \mu_{\mathcal{M}}\|^2 / 2\sigma^2) \tag{4}$$

其中:

$$b_{i|\mathcal{M}} = \{\frac{x_{b_i} - x_{\mathcal{M}}}{w_{\mathcal{M}}}, \frac{y_{b_i} - y_{\mathcal{M}}}{h_{\mathcal{M}}}, \log\frac{w_{b_i}}{w_{\mathcal{M}}}, \log\frac{h_{b_i}}{h_{\mathcal{M}}}\} \tag{5}$$

NOH-NMS 框架

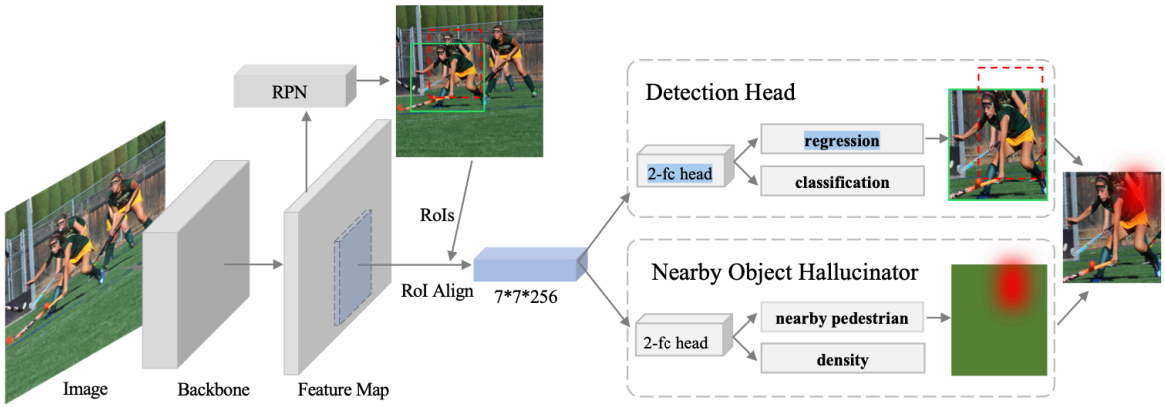

图 2: NOH-NMS 框架示意图

NOH-NMS Algorithm

input : $\mathcal{B} = b_1, \cdots, b_N, \mathcal{S} = s_1, \cdots, s_N, \mathcal{D} = d_1, \cdots, d_N, \mathcal{P} = p_1, \cdots, p_N, N_t$
$\mathcal{B}$ is the list of initial detection boxes, $\mathcal{S}$ contains corresponding detection scores, $\mathcal{D}$ contains corresponding detection densities, $\mathcal{P}$ contains the parameters of nearby-objects distribution of corresponding detection, N_t is the NMS threshold

begin
  $\mathcal{F} \leftarrow \{\}$;
  while $\mathcal{B} \neq \varnothing$ **do**
    $m \leftarrow \arg\max \mathcal{S}$;
    $\mathcal{M} \leftarrow b_m$;
    $\mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{M}; \mathcal{B} \leftarrow \mathcal{B} - \mathcal{M}$;
    for $b_i \in \mathcal{B}$ **do**
      if $\mathrm{iou}(\mathcal{M}, b_i) \geq N_t$ **then**
        $s_i \leftarrow s_i \cdot f(\mathcal{M}, b_i, d_{\mathcal{M}}, p_{\mathcal{M}})$;
      end
    end
  end
  return $\mathcal{F}, \mathcal{S}$
end

Implementation Details

- baseline: Faster-RCNN with FPN;

[1]Citypersons: A diverse dataset for pedestrian detection (CVPR 2017)

[2]Crowdhuman: A benchmark for detecting human in a crowd(arXiv preprint arXiv:1805.00123)

Implementation Details

- baseline: Faster-RCNN with FPN;
- datasets: CityPersons[1] 以及 CrowdHuman[2]

[1] Citypersons: A diverse dataset for pedestrian detection (CVPR 2017)

[2] Crowdhuman: A benchmark for detecting human in a crowd(arXiv preprint arXiv:1805.00123)

Implementation Details

- baseline: Faster-RCNN with FPN;
- datasets: CityPersons[1] 以及 CrowdHuman[2]
- ResNet-50 as the backbone and replace the ROIPooling operation in the original Faster-RCNN with the RoIAlign.

[1] Citypersons: A diverse dataset for pedestrian detection (CVPR 2017)

[2] Crowdhuman: A benchmark for detecting human in a crowd(arXiv preprint arXiv:1805.00123)

Results

Methods	Extra Anno.	Backbone	Scale	Reasonable	Bare	Partial	Heavy
OR-CNN [35]	✓	VGG-16	×1.3	11.0	**5.9**	**13.7**	51.3
MGAN [22]	✓	VGG-16	×1.3	10.5	-	-	**47.2**
JointDet [3]	✓	ResNet-50	×1.3	**10.2**	-	-	-
TLL (MRF) [29]		ResNet-50	-	14.4	-	-	-
Adapted Faster RCNN [34]		VGG-16	×1.3	13.0	-	-	-
ALFNet [21]		VGG-16	×1	12.0	8.4	11.4	**51.9**
RepLoss [31]		ResNet-50	×1.3	11.6	7.0	14.8	55.3
Adaptive-NMS w/ AggLoss [18]		VGG-16	×1.3	**10.8**	**6.2**	11.4	54.0
Our baseline		ResNet-50	×1.3	11.9	7.4	12.3	53.0
NOH-NMS		ResNet-50	×1.3	**10.8**	6.6	**11.2**	53.0

Table 1: Performance on the CityPersons validation set. MR^{-2} is used as the metric (lower is better). Scale is short for input scale.

Results

Methods	N_t	AP	Recall	MR^{-2}
Greedy-NMS	0.5	85.1	87.8	44.7
Soft-NMS [1]	0.5	86.4	90.6	44.6
Adaptive-NMS [18]	0.5	87.1	89.2	45.0
NOH-NMS	0.5	**89.0**	**92.9**	**43.9**

Table 3: Comparison of different NMS methods on the CrowdHuman validation set. All the methods are implemented by us, and for fair comparisons, we show the best results from multiple runs.

A Further Step

- NMS 在 YOLO v8 中的处理时间已经成为了瓶颈，过去的工作大多数是以提升准确率为目标，鲜有优化复杂度的工作；

A Further Step

- NMS 在 YOLO v8 中的处理时间已经成为了瓶颈，过去的工作大多数是以提升准确率为目标，鲜有优化复杂度的工作；
- Fast NMS 提供了一些思路，但是由于其需要计算 IoU 矩阵，复杂度仍为 $\mathcal{O}(N^2)$；

A Further Step

- NMS 在 YOLO v8 中的处理时间已经成为了瓶颈，过去的工作大多数是以提升准确率为目标，鲜有优化复杂度的工作；
- Fast NMS 提供了一些思路，但是由于其需要计算 IoU 矩阵，复杂度仍为 $\mathcal{O}(N^2)$；
- 本质是计算最大的前缀 iou 值；

A Further Step

- NMS 在 YOLO v8 中的处理时间已经成为了瓶颈，过去的工作大多数是以提升准确率为目标，鲜有优化复杂度的工作；
- Fast NMS 提供了一些思路，但是由于其需要计算 IoU 矩阵，复杂度仍为 $\mathcal{O}(N^2)$；
- 本质是计算最大的前缀 iou 值；
- 简化问题为询问中心点 k 近邻的矩形，online KD-Tree 或者基于随机的平面分割算法可以在 $\mathcal{O}(k\log^r(N))$ 的复杂度内查询。

Thanks!