

yolo-6d

论文地址：<https://arxiv.org/abs/1711.08848>

1 创新点

(1) 提出一个单阶段、仅需要RGB图像、可同时进行目标检测和6d姿态估计的快速网络，在Titan X GPU上可达到50fps

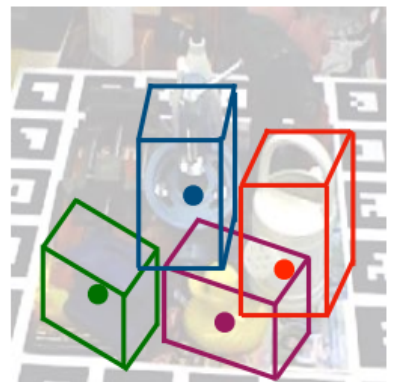
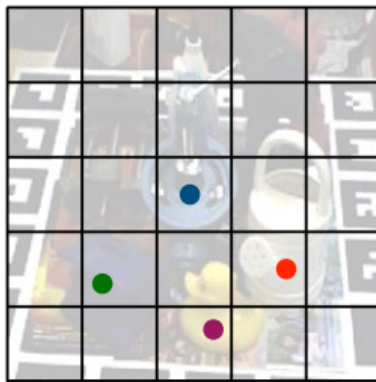
(2) 6d姿态估计的做法是在当前姿态下，网络预测3d物体模型边缘构成的8个坐标点和物体中心点一共9个点在2d图片上的投影坐标，最终使用n点透视PnP算法(Perspective n point)计算出预测6d姿态。

本文思想来源于论文BB8: A Scalable, Accurate, Robust to Partial Occlusion Method for Predicting the 3D Poses of Challenging Objects without Using Depth，由于使用yolo网络，使其具备非常快的预测速度，同时作者指出如果使用post-processing，可以实现更高的精度，但是速度会下降到10fps。

2 核心理想

BB8是第一篇将CNN应用于6d姿态估计的论文，效果非常显著，但是其不是一个端到端的算法，分成好几个阶段，并且和SSD-6D一样，需要pose refinement。基于以上问题，作者采用BB8论文思想，但是网络进行了修改，在速度提升的同时也提高了精度(速度和精度的提高都是因为yolo，作者本文没有大改进)。

BB8预测6d姿态的核心思想就是不直接预测3d物体表面上点到2d图片的映射点的关系，因为这样复杂度太高，而是预测3d物体所构成的3d边界框在2d图片上的投影坐标点，如果网络预测得到2d图片投影坐标点后，就可以使用PnP算法(PnP求解算法是指通过多对3D与2D匹配点，在已知或者未知相机内参的情况下，利用最小化重投影误差来求解相机外参的算法，PnP求解算法是SLAM前端位姿跟踪部分中常用的算法之一，opencv可以直接调用api)直接得到3d旋转矩阵和3d平移向量。然而本文采用的是9个点，多预测一个物体中心点(我猜想是因为yolo网络的特点，需要一个中心点，而不是说有了中心点，重建的3d姿态会更准)。如下图所示：



3 模型

3.1 模型结构

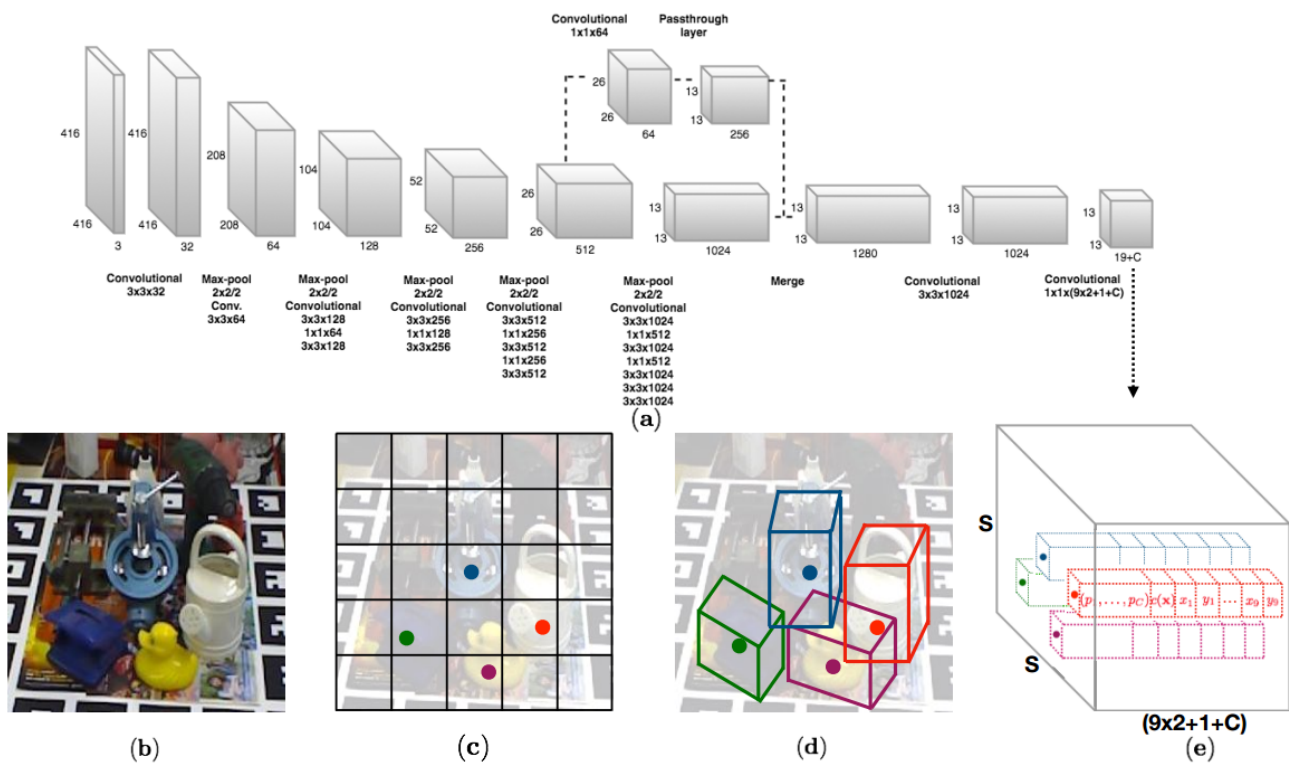


图 yolo-6d网络结构

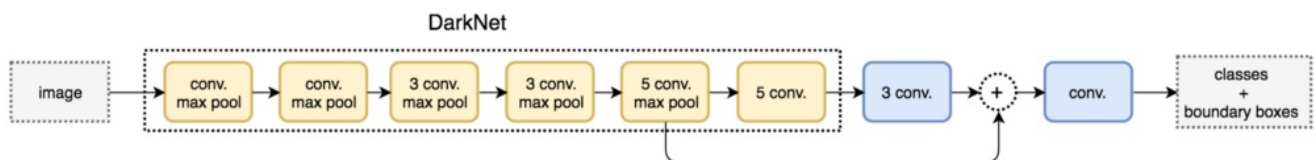


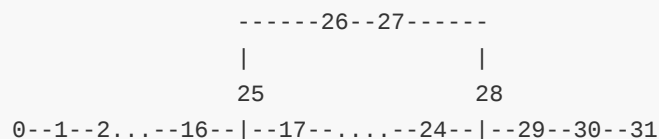
图 yolo-v2网络结构

本文整个网络结构是yolo-v2，如上图所示，但是训练策略不是完全按照yolo-v2的，而是作者自己设计的，后面会细说。yolo-v2是由基础网络darkNet-19和后续的头网络构成，基础网络包括19个卷积+5个最大池化层，通过passthrough方式将底层特征图直接快捷连接到高层特征图，由于底层特征图的高分辨率信息的融合，有利于目标检测。本文yolo-6d网络结构和yolo-v2几乎完全相同，唯一区别是最后一个卷积层的通道输出不同，yolo-v2是输出classes+bbbox，而yolo-6d是输出classes+9个控制点坐标。

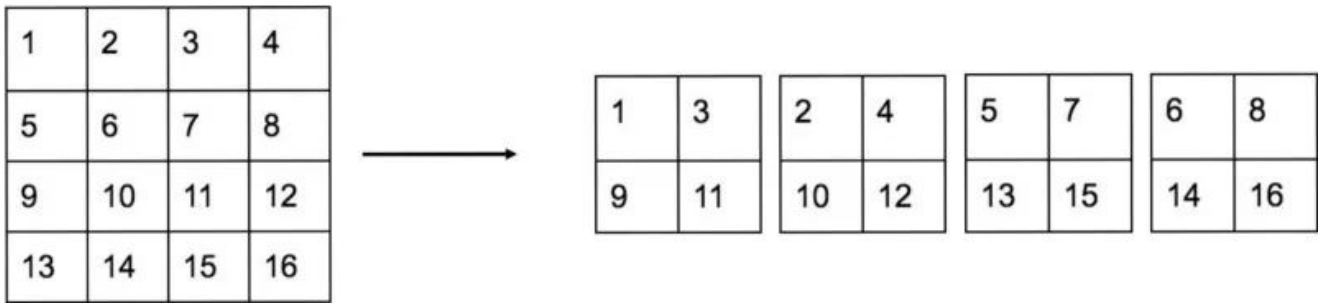
基于作者已经开源的pytorch代码，其设计的网络整体结构如下：

layer	filters	size	input	output
0 conv	32	3 x 3 / 1	416 x 416 x 3	416 x 416 x 32
1 max		2 x 2 / 2	416 x 416 x 32	208 x 208 x 32
2 conv	64	3 x 3 / 1	208 x 208 x 32	208 x 208 x 64
3 max		2 x 2 / 2	208 x 208 x 64	104 x 104 x 64
4 conv	128	3 x 3 / 1	104 x 104 x 64	104 x 104 x 128
5 conv	64	1 x 1 / 1	104 x 104 x 128	104 x 104 x 64
6 conv	128	3 x 3 / 1	104 x 104 x 64	104 x 104 x 128
7 max		2 x 2 / 2	104 x 104 x 128	52 x 52 x 128
8 conv	256	3 x 3 / 1	52 x 52 x 128	52 x 52 x 256
9 conv	128	1 x 1 / 1	52 x 52 x 256	52 x 52 x 128
10 conv	256	3 x 3 / 1	52 x 52 x 128	52 x 52 x 256
11 max		2 x 2 / 2	52 x 52 x 256	26 x 26 x 256
12 conv	512	3 x 3 / 1	26 x 26 x 256	26 x 26 x 512
13 conv	256	1 x 1 / 1	26 x 26 x 512	26 x 26 x 256
14 conv	512	3 x 3 / 1	26 x 26 x 256	26 x 26 x 512
15 conv	256	1 x 1 / 1	26 x 26 x 512	26 x 26 x 256
16 conv	512	3 x 3 / 1	26 x 26 x 256	26 x 26 x 512
17 max		2 x 2 / 2	26 x 26 x 512	13 x 13 x 512
18 conv	1024	3 x 3 / 1	13 x 13 x 512	13 x 13 x 1024
19 conv	512	1 x 1 / 1	13 x 13 x 1024	13 x 13 x 512
20 conv	1024	3 x 3 / 1	13 x 13 x 512	13 x 13 x 1024
21 conv	512	1 x 1 / 1	13 x 13 x 1024	13 x 13 x 512
22 conv	1024	3 x 3 / 1	13 x 13 x 512	13 x 13 x 1024
23 conv	1024	3 x 3 / 1	13 x 13 x 1024	13 x 13 x 1024
24 conv	1024	3 x 3 / 1	13 x 13 x 1024	13 x 13 x 1024
25 route	16			
26 conv	64	1 x 1 / 1	26 x 26 x 512	26 x 26 x 64
27 reorg		/ 2	26 x 26 x 64	13 x 13 x 256
28 route	27 24			
29 conv	1024	3 x 3 / 1	13 x 13 x 1280	13 x 13 x 1024
30 conv	20	1 x 1 / 1	13 x 13 x 1024	13 x 13 x 20
31 detection				

网络连接结构如下：



序号0~22为darknet-19基础网络部分，其中1,3,7,11,17为池化层，其他为卷积层。为了引入高分辨率特征信息，passthrough操作，具体是：25和28为快捷连接，26为1x1的卷积核，目的是减少通道，27为reorg特征重排层，前面的特征图维度(26 x 26 x 64)是后面的特征图(13 x 13 x 1024)的2倍，passthrough层抽取前面层的每个2*2的局部区域，然后将其转化为channel维度，对于(26 x 26 x 64)的特征图，经passthrough层处理之后就变成了(13 x 13 x 256)的新特征图，示意图如下：



第28个快捷连接层输出维度是(13 x 13 x256)，然后和第24个卷积层(13 x 13 x1024)，进行拼接操作，得到(13x13x1280)。29,30是卷积层，第31层是loss层。所以整个yolo-6d网络总共31层。

假设网络将图片划分为 $S * S$ 个网格，那么输出维度是 $S * S * D$ ， D 是 $9 \times 2 + 1 + C$ ，其中9是2d图片上的9个控制点， x_2 是x和y坐标， C 是物体类别概率，1是confidence置信度，非常容易理解。因为很有可能多个物体靠的很近，此时一个cell预测一组2d坐标值肯定是不够的，故yolo-v2引入了anchor机制，同样yolo-6d也引入了anchor，所以作者网络的实际输出并不是上面那个，而是 $S * S * [\text{anchor_num} * (9 * 2 + 1 + C)]$ ，作者设置的 $\text{anchor_num}=5$ ，5种anchor的设置和yolo-v2一样，是采用k-mean确定。而 S 的值也不是固定的，需要根据输入图片的宽高实时计算，即最后一层特征图的宽高是多少，那么 S 就设置为该值即可，其实我们可以提前计算出来，因为最后一层特征图的宽高就是输入图片宽高的1/32，如果输入是416x416，那么 $S=13$ 。

3.2 网络处理

在yolo-v2中，任何一个网格cell，输出的是bbox值和confidence置信度，其中confidence表示cell含有物体的概率以及bbox的准确度($\text{confidence} = P(\text{object}) * \text{IOU}_{\text{pred}}^{\text{truth}}$)，由于本文算法没有直接预测bbox所以无法计算IOU，所以需要进行修改，如下图所示。

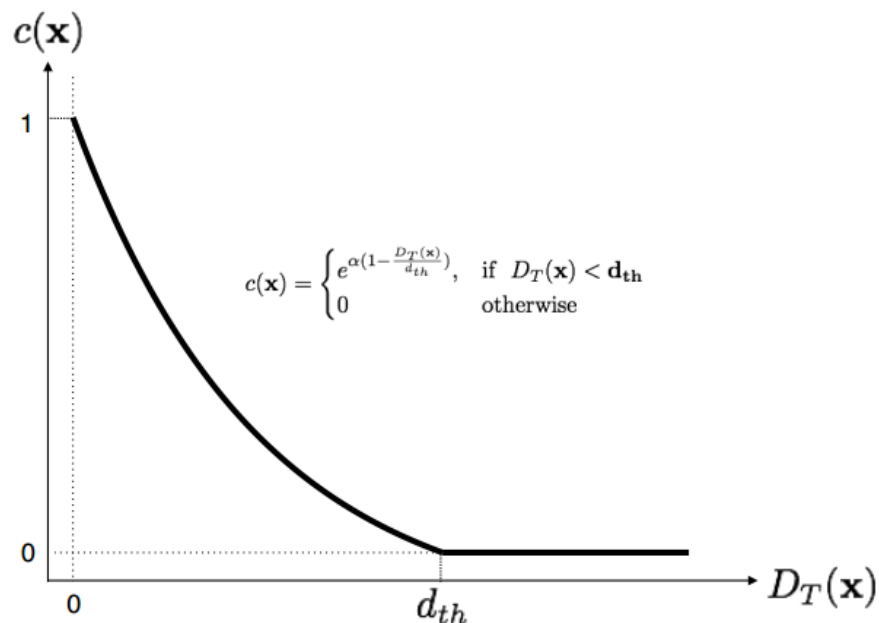


Figure 2. Confidence $c(x)$ as a function of the distance $D_T(x)$ between a predicted point and the true point.

Confidence修改为 $c(x)$ ， $D_T(x)$ 是预测的坐标值与真实值之间的欧式距离， d_t 是提前设定的阈值30px，最终的 $c(x)$ 是9个点的Confidence取均值， α 是超参，作者设置为2。从上图可以看出，当预测值与真实值越接近时候， $c(x)$ 值越大。但是仔细看公式，可以发现公式和图性是对不上的。实际上公式应该是：

$$(e^{\alpha(1-D_T(x)/d_{th}} - 1)/(e^{\alpha} - 1), \text{ if } D_T(x) < d_{th}$$

4 训练

4.1 训练策略

网络的最后一层的输出是 $S * S * [anchor_num * (9 * 2 + 1 + C)]$, 对于每个cell grid中的每个anchor输出物体所属类别概率值、9个控制点的平均confidence值和9个(x,y)坐标值。对于(x,y)坐标值需要特别注意：因为预测中心点一定在网格内，所以2d物体中心点坐标的输出预测值需要约束到0~1之间，但是其余8个点坐标是不能约束的，因为很可能在当前网格的外面，故具体实现是：

$$g_x = f(x) + c_x \quad (2)$$

$$g_y = f(y) + c_y \quad (3)$$

对于中心点， $f()$ 取sigmoid函数，映射到0~1之间，对于其他8个点， $f(x) = x$ 即可， c_x 和 c_y 是当前网格的归一化坐标值(例如网格大小是13x13，则(0,1)坐标处的cell的 c_x 和 c_y 就是(0,1))，这个设定和yolo-v2是一样的。

yolo-v2采用了多分辨率图片输入进行训练，yolo-6d也是一样，具体是从{320,352,...,608}集合中随机选择分辨率大小的图片进行训练，它们都是32的倍数，输入分辨率改变了，那么最后一层的输出网格个数也会随之改变，在计算ground truth时候也要改变维度。

4.2 损失函数

损失函数定义为：

$$\mathcal{L} = \lambda_{pt} \mathcal{L}_{pt} + \lambda_{conf} \mathcal{L}_{conf} + \lambda_{id} \mathcal{L}_{id} \quad (4)$$

pt代表9个坐标点，conf代表置信度，id代表类别，前两个损失函数是均分误差，最后一个损失函数交叉熵。对于包括物体的网格，conf项loss的权重是5，不包括物体是0.1，其他loss权重设为1。可以看出这个loss其实和yolo-v1很类似，和yolo-v2是不同的(yolo-v2的损失函数定义非常复杂)。这个loss写的过于简洁了，为了方便理解，下面给出yolo-v1和v2的损失函数定义：

$$\begin{aligned} & \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\ & + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\ & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 \\ & + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2 \\ & + \sum_{i=0}^{S^2} \mathbb{1}_i^{obj} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \quad (3) \end{aligned}$$

图 yolo-v1的损失定义

confidence和类别的loss设置和yolo-v1是完全一样的，但是由于yolo-6d没有bbox，所以只有 x_i ，而没有 w_i 部分的loss，并且由于引入了anchor，所以以上所有项都有再加上一层anchor的循环，就像下面的yolo-v2 loss一样。

说到anchor，就必须说到anchor匹配规则，由于yolo-v2和yolo-6d论文都没有说这部分，参考其他资料应该是：对于某个ground truth，首先要确定其中心点要落在哪个cell上，然后计算这个cell的5个先验框与ground truth的IOU值，计算IOU值时不考虑坐标，只考虑形状，所以先将先验框与ground truth的中心点都偏移到同一位置（原点），然后计算出对应的IOU值，IOU值最大的那个先验框与ground truth匹配，对应的预测框用来预测这个ground truth；如果一个cell里面同时存在多个物体中心，那么其他由其他剩下的先验框进行匹配，也就是说一个cell内部最多预测5个物体，实际上肯定是够了。

$$\begin{aligned}
 loss_t = & \sum_{i=0}^W \sum_{j=0}^H \sum_{k=0}^A 1_{Max IOU < Thresh} \lambda_{noobj} * (-b_{ijk}^o)^2 \\
 & + 1_{t < 12800} \lambda_{prior} * \sum_{r \in (x,y,w,h)} (prior_k^r - b_{ijk}^r)^2 \\
 & + 1_k^{truth} (\lambda_{coord} * \sum_{r \in (x,y,w,h)} (truth^r - b_{ijk}^r)^2 \\
 & \quad + \lambda_{obj} * (IOU_{truth}^k - b_{ijk}^o)^2 \\
 & \quad + \lambda_{class} * (\sum_{c=1}^C (truth^c - b_{ijk}^c)^2))
 \end{aligned}$$

图 yolo-v2的loss定义

在前向预测阶段，考虑到物体有大有小、有密集有稀疏，很可能出现多个网格同时输出非常高的置信度，为了结果更加鲁棒，我们对每个cell，要关注周边的8个cell，首先选择周边的8个cell，并对周边的每个cell(预测5组输出值)都取最大置信度所对于的预测值，然后基于每个cell的置信度取值大小进行加权平均，得到最终的预测值。具体是：对于每个cell预测得到的其中一个坐标点，首先选取周边的8个cell，都取最大置信度所对于的输出坐标预测值，然后利用当前cell的坐标预测值等于所有9个cell预测的坐标值进行加权平均，权重就是对应的置信度。

在最终得到9个2d坐标值后，使用PnP算法即可得到3d旋转矩阵和3d平移向量。

具体网络训练策略是：网络的初始化参数是使用ImageNet分类任务训练后的预训练权重进行初始化即darknet预训练权重。考虑到前期网络的6d估计值非常不稳定，难以训练，所以我们通过将置信度loss设置为0来预训练我们的网络参数。然后基于当前得到的预训练权重再次训练网络，此时的置信度Loss就是正常训练，上述训练方式作者通过实验发现效果更好。其他网络参数设置为SGD，初始学习率为0.001，每过100个回合除以10，为了避免过拟合，使用了数据增强操作，分别是色调、饱和度和曝光、随机缩放、翻转，这些设置和yolo-v1一致。和yolo-v2不同的是训练数据不是直接使用原始linemod数据，而是随机选择voc2012数据作为背景，替换掉原始的linemod背景，作者指出目的是防止过拟合。

通过作者的开源pytorch代码发现：当训练数据是单目标时候，网络训练策略非常像yolo-v1，其没有引入anchor，或者说anchor是1；当训练数据是多目标时候，训练策略就是和yolo-v2一致。

5 结果

实验在两个数据集上面测试，分别是LineMod数据集和OCCLUSION LineMod数据集。

Method	w/o Refinement			w/ Refinement	
Object	Brachmann [2]	BB8 [26]	OURS	Brachmann [2]	BB8 [26]
Ape	-	95.3	92.10	85.2	96.6
Benchvise	-	80.0	95.06	67.9	90.1
Cam	-	80.9	93.24	58.7	86.0
Can	-	84.1	97.44	70.8	91.2
Cat	-	97.0	97.41	84.2	98.8
Driller	-	74.1	79.41	73.9	80.9
Duck	-	81.2	94.65	73.1	92.2
Eggbox	-	87.9	90.33	83.1	91.0
Glue	-	89.0	96.53	74.2	92.3
Holepuncher	-	90.5	92.86	78.9	95.3
Iron	-	78.9	82.94	83.6	84.8
Lamp	-	74.4	76.87	64.0	75.8
Phone	-	77.6	86.07	60.6	85.3
Average	69.5	83.9	90.37	73.7	89.3

图 linemod数据集上的2d pose指标

Method	w/o Refinement				w/ Refinement		
Object	Brachmann [2]	BB8 [26]	SSD-6D [11]	OURS	Brachmann [2]	BB8 [26]	SSD-6D [11]
Ape	-	27.9	0	21.62	33.2	40.4	65
Benchvise	-	62.0	0.18	81.80	64.8	91.8	80
Cam	-	40.1	0.41	36.57	38.4	55.7	78
Can	-	48.1	1.35	68.80	62.9	64.1	86
Cat	-	45.2	0.51	41.82	42.7	62.6	70
Driller	-	58.6	2.58	63.51	61.9	74.4	73
Duck	-	32.8	0	27.23	30.2	44.3	66
Eggbox	-	40.0	8.9	69.58	49.9	57.8	100
Glue	-	27.0	0	80.02	31.2	41.2	100
Holepuncher	-	42.4	0.30	42.63	52.8	67.2	49
Iron	-	67.0	8.86	74.97	80.0	84.7	78
Lamp	-	39.9	8.20	71.11	67.0	76.5	73
Phone	-	35.2	0.18	47.74	38.1	54.0	79
Average	32.3	43.6	2.42	55.95	50.2	62.7	79

Table 2. Comparison of our approach with state-of-the-art algorithms on LINEMOD in terms of ADD metric. We report percentages of correctly estimated poses.

Method	MAP
Hinterstoisser et al. [8]	0.21
Brachmann et al. [2]	0.51
Kehl et al. [11]	0.38
OURS	0.48

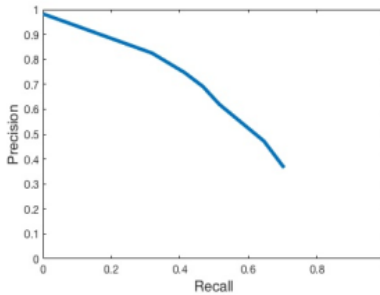


Table 6. The detection experiment on the Occlusion dataset [2]. (Left) Precision-recall plot. (Right)

MAP是mean average precision。

Resolution	2D projection metric	Speed
416×416	89.71	94 fps
480×480	90.00	67 fps
544×544	90.37	50 fps
688×688	90.65	43 fps

Table 7. Accuracy/speed trade-off of our method on the LINEMOD dataset. Accuracy reported is the percentage of correctly estimated poses w.r.t the 2D projection error. The same network model is used for all four input resolutions. Timings are on a Titan X (Pascal) GPU.

