CornerNet: Detecting Objects as Paired Keypoints

Hei Law · Jia Deng

Abstract

우리는 object bounding box를 한 쌍의 keypoints(좌측 상단과 우측 하단 corner)로 감지하는 새로운 object detection 방식인 CornerNet을 single CNN을 사용하여 제안한다. 쌍으로 구성된 keypoints로 객체를 검출함으로써, 우리는 이전의 one-stage detector에서 일반적으로 사용되는 anchor box 세트를 디자인할 필요성을 제거한다. 우리의 novel한 공식 뿐만 아니라, 우리는 네트워크가 corner를 더 잘 localize하는데 도움을 주는 새로운 유형의 pooling layer인 corner pooling을 소개한다. 실험에 따르면 CornerNet은 MS COCO에서 42.2% AP를 달성하여, 기존의 모든 one-stage detector보다 우수한 성능을 보였다.

1. Introduction

CNN 기반 객체 detector는 다양한 challenging한 벤치마크에서 state-of-the-art한 결과를 얻었다. state-of-the-art한 접근법의 공통적인 구성요소는 anchor box인데, 다양한 크기와 aspect ratio를 가진 box로서 검출 후보 역할을 한다. 앵커 박스는 one-stage detector에 광범위하게 사용되며, two-stage detector와 높은 경쟁력을 달성하는 동시에 효율을 높일 수 있다. One-stage detector는 anchor box를 이미지 위에 dense하게 배치하고 anchor box를 scoring하고 regression을 통해 좌표를 조정하여 최종 box prediction을 생성한다.

그러나 anchor box의 사용은 두 가지 단점이 있다. 첫째, 우리는 일반적으로 매우 큰 anchor box가 필요하다(DSSD에서는 40k 이상, RetinaNet에서는 100k 이상). 이는 detector가 각각의 anchor box가 ground truth box와 충분히 중복되는지 여부를 분류하도록 훈련되고, 대부분의 ground truth box와 충분한 overlap이 보장되도록 많은 수의 anchor box가 필요하기 때문이다. 그 결과, anchor box의 아주 작은 부분만이 ground truth box와 overlap될 것이다. 이것은 positive/negative anchor box 간의 엄청난 불균형을 만들고 학습을 지연시킨다.

둘째로, anchor box의 사용은 많은 hyperparameter와 디자인 선택을 도입한다. 여기에는 박스 수, 크기, aspect ratio가 포함된다. 그러한 선택은 주로 ad-hoc heuristics를 통해 이루어졌으며, 단일 네트워크가 여러 해상도에서 별도의 prediction을 하는 multiscale architecture와 결합하면 훨씬 더 복잡해질 수 있으며, 각 scale은 서로 다른 feature과 그 자체의 anchor box 세트를 사용한다.

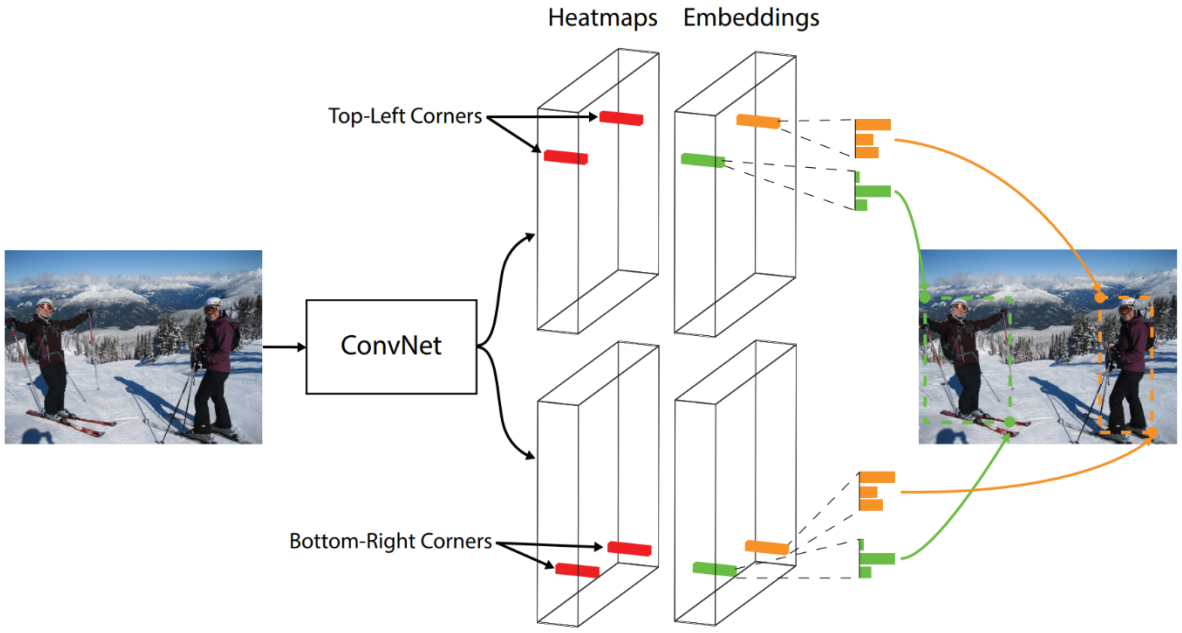

그림 1 우리는 한 쌍의 bounding box corner가 함께 그룹화한 것을 감지한다. ConvNet은 모든 좌측 상단 / 우측 하단 corner에 대한 heatmap 및 검출된 각 corner에 대한 embedding vector를 출력한다. 네트워크는 동일한 객체에 속하는 corner에 대해 유사한 embeddings를 predict하도록 학습된다.

본 논문에서 우리는 anchor box를 없애주는 객체 검출에 대한 새로운 one-stage 접근법인 CornerNet을 소개한다. 우리는 bounding box의 좌측 상단 corner와 우측 하단 corner에 있는 한 쌍의 keypoint로 객체를 검출한다. Single CNN을 사용하여 동일한 객체 카테고리의 모든 instance의 좌측 상단 / 우측 하단 모서리에 대한 heatmap 및 검출된 corner에 대한 embedding vector를 predict한다. 이 embedding은 같은 객체에 속하는 한 쌍의 corner를 그룹화하는 역할을 한다. 네트워크는 이들을 위한 유사한 embedding을 predict하도록 학습한다. 우리의 접근방식은 네트워크의 출력을 크게 단순화하고 anchor box 설계의 필요성을 제거한다. 또한, multi-person human-pose estimation의 맥락에서 keypoint를 검출하고 그룹화하는 Newell이 제안한 associative embedding 방법에 영감을 받았다. 그림 1은 우리의 접근방식의 전체적인 파이프라인을 보여준다.

그림 2 흔히 bounding box corner의 위치를 결정하는 local evidence가 없다. 우리는 새로운 유형의 pooling layer를 제안함으로써 이 문제를 해결한다.

CornerNet의 또다른 새로운 구성요소는 corner pooling인데, 이것은 복잡한 네트워크가 bounding box corner를 더 잘 localize하도록 돕는 새로운 유형의 pooling layer이다. Bounding box corner는 종종 객체 바깥에 있다. 그림 2의 예와 함께 원의 경우를 보자. 각 케이스에서 corner는 local evidence에 근거하여 localize할 수 없다. 대신 픽셀 위치에 좌측 상단 corner가 있는지 확인하기 위해서는 객체의 맨 위 경계를 우측을 향해 수평으로 바라보고, 맨 왼쪽 경계를 위해 수직으로 아래쪽을 보아야 한다. 이것은 우리의 corner pooling layer를 motivate한다. 두 개의 feature map을 가지고 있는데, 각 픽셀 위치에서 첫 번째 feature map에서 모든 feature vector를 오른쪽으로 max pooling하고, 두 번째 feature map에서 바로 아래에 모든 feature vector를 max pooling한 다음 두 개의 결과를 함께 추가한다. 그림 3을 보자.

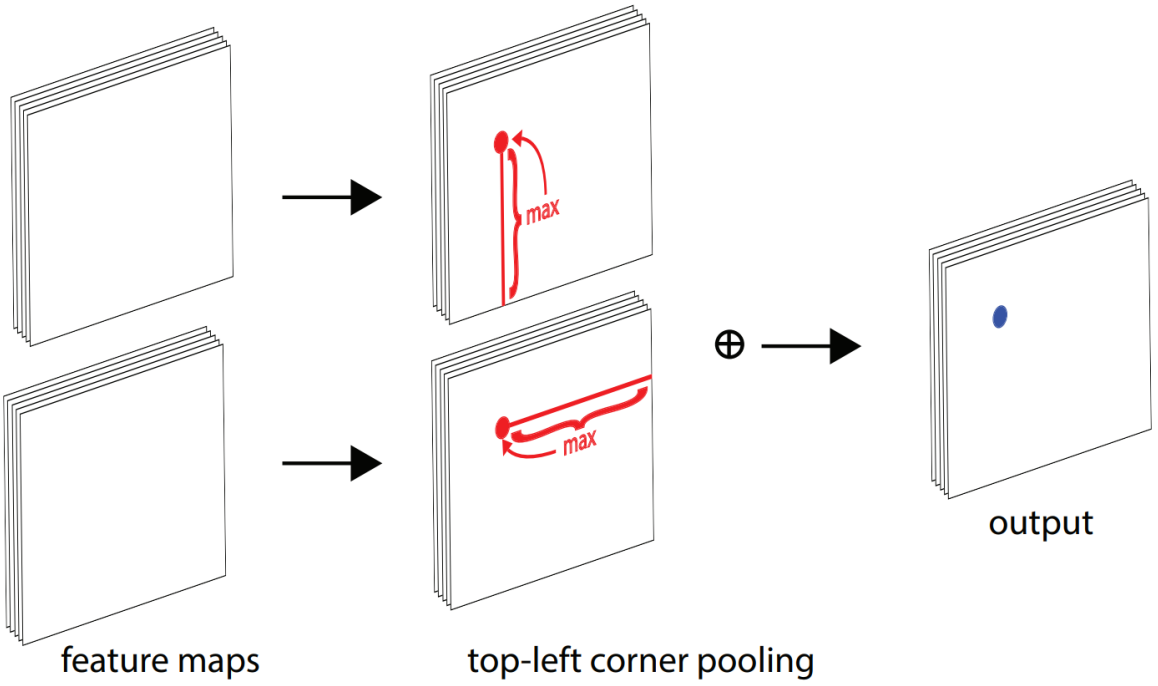

그림 3 Corner pooling: 각 채널에 대해, 각각 별도의 feature map에서 최대값(빨간색 점)을 두 방향(빨간색 선)으로 취하여 두 개의 최대값(파란색 점)을 함께 추가한다

우리는 왜 corner를 검출하는 것이 box 중심이나 제안보다 더 잘 작동하는지 두 가지 이유를 가정한다. 첫째, box의 중심은 localize하기 어려울 수 있는데, 객체의 4면 모두에 의존하는 반면, corner를 locating하는 것은 2개 면에 의존하기 때문에 더 쉽다. 그리고 corner pooling의 경우 더욱 그러하며, 이는 corner의 정의에 대한 명확한 사전 지식을 인코딩한다. 둘째, corner는 box 공간을 densely 하게 구분하는 더 효율적인 방법을 제공한다. 우리는 단지 O(w^2h^2)의 anchor box를 나타내기 위해서 O(wh)의 코너만 있으면 된다.

우리는 MS COCO에 대한 CornerNet의 효과를 입증한다. CornerNet은 기존의 모든 one-stage 검출기를 능가하는 42.1% AP를 달성했다. 또한 ablation study를 통해 corner pooling이 CornerNet의 우수한 성능에 매우 중요하다는 것을 알 수 있다. 코드는 https://github.com/umichvl/CornerNet 에서 이용 가능하다.

2. Related Works

Two-stage object detectors

Two-stage approach was first introduced and popularized by R-CNN [12]. Two-stage detectors generate a sparse set of regions of interest (RoIs) and classify each of them by a network. R-CNN generates RoIs using a low level vision algorithm [41, 47]. Each region is then extracted from the image and processed by a ConvNet independently, which creates lots of redundant computations. Later, SPP [14] and Fast-RCNN [11] improve R-CNN by designing a special pooling layer that pools each region from feature maps instead. However, both still rely on separate proposal algorithms and cannot be trained end-to-end. Faster-RCNN [32] does away low level proposal algorithms by introducing a region proposal network (RPN), which generates proposals from a set of pre-determined candidate boxes, usually known as anchor boxes. This not only makes the detectors more efficient but also allows the detectors to be trained end-to-end. R-FCN [6] further improves the efficiency of Faster-RCNN by replacing the fully connected sub-detection network with a fully convolutional sub-detection network. Other works focus on incorporating sub-category information [42], generating object proposals at multiple scales with more contextual information [1, 3, 35, 22], selecting better features [44], improving speed [21], cascade procedure [4] and better training procedure [37].

One-stage object detectors

On the other hand, YOLO [30] and SSD [25] have popularized the one-stage approach, which removes the RoI pooling step and detects objects in a single network. One-stage detectors are usually more computationally efficient than two-stage detectors while maintaining competitive performance on different challenging benchmarks.

SSD places anchor boxes densely over feature maps from multiple scales, directly classifies and refines each anchor box. YOLO predicts bounding box coordinates directly from an image, and is later improved in YOLO9000 [31] by switching to anchor boxes. DSSD [10] and RON [19] adopt networks similar to the hourglass network [28], enabling them to combine low-level and high-level features via skip connections to predict bounding boxes more accurately. However, these one-stage detectors are still outperformed by the two-stage detectors until the introduction of RetinaNet [23]. In [23], the authors suggest that the dense anchor boxes create a huge imbalance between positive and negative anchor boxes during training. This imbalance causes the training to be inefficient and hence the performance to be suboptimal. They propose a new loss, Focal Loss, to dynamically adjust the weights of each anchor box and show that their one-stage detector can outperform the two-stage detectors. RefineDet [45] proposes to filter the anchor boxes to reduce the number of negative boxes, and to coarsely adjust the anchor boxes.

DeNet [39] is a two-stage detector which generates RoIs without using anchor boxes. It first determines how likely each location belongs to either the topleft, top-right, bottom-left or bottom-right corner of a bounding box. It then generates RoIs by enumerating all possible corner combinations, and follows the standard two-stage approach to classify each RoI. Our approach is very different from DeNet. First, DeNet does not identify if two corners are from the same objects and relies on a sub-detection network to reject poor RoIs. In contrast, our approach is a one-stage approach which detects and groups the corners using a single ConvNet. Second, DeNet selects features at manually determined locations relative to a region for classification, while our approach does not require any feature selection step. Third, we introduce corner pooling, a novel type of layer to enhance corner detection.

Our approach is inspired by Newell et al. work [27] on Associative Embedding in the context of multi-person pose estimation. Newell et al. propose an approach that detects and groups human joints in a single network. In their approach each detected human joint has an embedding vector. The joints are grouped based on the distances between their embeddings. To the best of our knowledge, we are the first to formulate the task of object detection as a task of detecting and grouping corners simultaneously. Another novelty of ours is the corner pooling layers that help better localize the corners. We also significantly modify the hourglass architecture and add our novel variant of focal loss [23] to help better train the network.

3. CornerNet

3.1. Overview

CornerNet에서는 bounding box의 좌측 상단 / 우측 하단 corner의 keypoint 쌍으로 객체를 검출한다. CNN은 두 heatmap 세트를 예측하여 서로 다른 객체 카테고리의 corner의 위치를 나타낸다. 하나는 좌측 상단, 다른 하나는 우측 하단 corner에 대해 설정된 것이다. 또한 동일한 객체에서 두 corner의 embedding 사이의 거리가 작도록, 네트워크는 검출된 각 corner에 대한 embedding vector를 predict한다. 더 타이트한 bounding box를 만들기 위해, 네트워크는 corner의 위치를 약간 조정하기 위해 offset을 predict한다. Predict된 heatmap, embedding, offset을 이용하여 간단한 후처리 알고리즘을 적용하여 최종적으로 bounding box를 구한다.

그림 4는 CornerNet의 개요를 보여준다. 우리는 CornerNet의 backbone 네트워크로 hourglass 네트워크를 사용한다. Hourglass에는 두 개의 prediction 모듈이 뒤따른다. 하나는 좌측 상단, 다른 하나는 우측 하단 corner를 위한 것이다. 각 모듈에는 heatmap, embedding, offset을 predict하기 전에 hourglass에서 feature를 pooling할 수 있는 자체 corner pooling 모듈이 있다. 다른 많은 객체 검출기와 달리, 우리는 다른 크기의 물체를 감지하기 위해 다른 scale의 feature를 사용하지 않는다. 우리는 hourglass의 output에만 두 모듈을 적용한다.

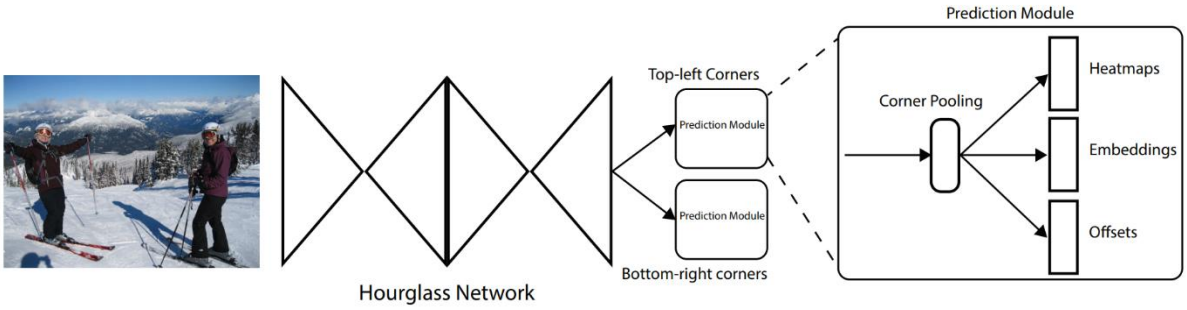

그림 4 CornerNet의 개요. Backbone 네트워크는 두 prediction 모듈이 뒤따른다. 하나는 좌측 상단, 다른 하나는 우측 하단 corner에 대한 것이다. 두 모듈의 prediction을 사용해서, corner를 찾아 그룹화한다.

3.2. Corner 검출

우리는 두 heatmap을 predict하는데, 하나는 좌측 상단, 다른 하나는 우측 하단 corner에 각각 하나씩 예측하고 있다. 각 heatmap 세트에는 카테고리 수 C 만큼의 채널이 있으며, 크기는 H x W 이다. 여기엔 background 채널이 없다. 각 채널은 클래스에 대한 corner의 위치를 나타내는 binary mask다.

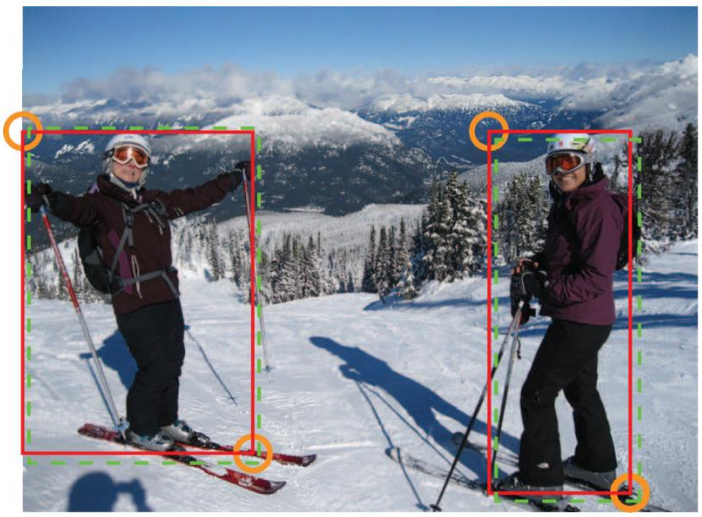

그림 5 학습을 위한 ground-truth heatmap. Corner가 positive한 위치(주황색 원)의 반경 내에 있는 박스(녹색 점선 직사각형)는 여전히 ground-truth annotation(빨간색 실선)과 큰 overlap을 가진다.

각 corner에 대해 하나의 ground-truth positive 위치가 있으며, 다른 모든 위치는 negative이다. 학습 중에, negative한 위치들을 똑같이 패널티를 주는 대신에, 우리는 positive 위치 반경 내의 negative에게 주어지는 패널티를 줄인다. 이는 한 쌍의 false corner 검출기 때문인데, 만약 그것들이 그들 각각의 ground-truth 위치에 가까우면, 여전히 ground-truth 박스와 충분히 overlap되는 박스를 만들 수 있기 때문이다(그림 5). 우리는 반경 내의 한 쌍의 점들이 적어도 t IoU가 있는 bounding box와 ground-truth annotation을 생성하도록 하여 반경을 객체의 크기로 결정한다 (t=0.7). 반경을 고려할 때, 패널티 감소량은 정규화되지 않은 2D Gaussian에 의해 주어지는데, 그 중심은 positive 위치에 있고 σ는 반경의 1/3이다.

p_{cij}는 predicted heatmap에서 클래스 c에 대한 위치 (i,j) 의 점수, y_{cij}는 정규화되지 않은 Gaussian과 함께 augmented된 ground-truth heatmap이라고 하자. focal loss의 variant를 설계하면,

$$L_{det} = \frac{-1}{N} \sum_{c=1}^{C} \sum_{i=1}^{H} \sum_{j=1}^{W} \begin{cases} (1-p_{cij})^{\alpha} \log(p_{cij}) & \text{if } y_{cij} = 1 \\ (1-y_{cij})^{\beta} (p_{cij})^{\alpha} \log(1-p_{cij}) & \text{otherwise} \end{cases}$$

여기서 N은 이미지의 객체 수이고, α와 β는 각 점의 contribution을 제어하는 hypermarameter 이다($\alpha = 2, \beta = 4$). y_{cji}로 인코딩된 Gaussian bump를 사용하면 $(1-y_{cij})$는 ground-truth 위치 주변의 패널티를 감소시킨다.

많은 네트워크들은 global한 정보를 수집하고 메모리 사용을 줄이기 위해 다운샘플링 layer를포함한다. 그것들이 완전히 convolutional하게 이미지에 적용될 때, 일반적으로 output 크기는 이미지보다 작다. 따라서 이미지의 위치 (x,y) 는 heatmap의 위치 $(\lfloor \frac{x}{n} \rfloor, \lfloor \frac{y}{n} \rfloor)$ 에 매핑된다(n은 다운샘플링 계수). Heatmap 에서 입력 이미지로 위치를 re-map하면 어느 정도의 precision이 손실될 수 있으며, 이는 ground-truth로 작은 bounding box의 IOU에 큰 영향을 미칠 수 있다. 이 문제를 해결하기 위해, 우리는 입력 resolution으로 re-mapping하기 전에 corner 위치를 약간 조정하기 위해 위치 offset을 predict한다.

$$\boldsymbol{o}_k = \left(\frac{x_k}{n} - \left\lfloor \frac{x_k}{n} \right\rfloor, \frac{y_k}{n} - \left\lfloor \frac{y_k}{n} \right\rfloor \right)$$

여기서 O_k는 offset이고, x_k와 y_k는 corner k의 좌표이다. 특히, 우리는 모든 카테고리의 좌측 상단 / 우측 하단 corner가 공유하는 offset 세트를 predict한다. 학습을 위해, 우리는 ground-truth corner 위치에 smooth한 L1 Loss를 적용한다.

$$L_{off} = \frac{1}{N} \sum_{k=1}^{N} \text{SmoothL1Loss}(\boldsymbol{o}_k, \hat{\boldsymbol{o}}_k)$$

3.3. Corner 그룹화

이미지에 여러 객체가 나타날 수 있으므로, 여러 개의 corner가 감지될 수 있다. 우리는 좌측 상단과 우측 하단 corner 한 쌍이 같은 bounding box에서 나왔는지 여부를 판단할 필요가 있다. 우리의 접근법은 Newell et al이 multi-person pose estimation 과제에 대해 제안한 Associative Embedding 방법에서 영감을 얻었다. 그들은 모든 인체 관절을 검출하고 검출된 각 관절에 대해 embedding을 생성한다. 그들은 embedding 사이의 거리를 기준으로 관절을 분류한다.

Associative Embedding이라는 아이디어는 우리의 과제에도 적용된다. 네트워크는 "corner들이 동일한 bounding box에 속할 경우, embedding 사이의 거리가 작아야 한다"고 검출된 각 corner에 대한 embedding vector를 예측한다. 그 다음 두 corner간의 embedding 사이의 거리를 기준으로 corner를 그룹화할 수 있다. Embedding 값은 중요하지 않다. 거리만이 corner를 묶는데 사용된다

우리는 Newell et al을 따르고 1D embedding을 사용한다. e_{t_k}를 객체 k의 좌측 상단(top), e_{b_k}를 우측 하단(bottom) corner에 대한 embedding이라고 하자. 우리는 corner를 그룹화하도록 네트워크를 학습할 "Pull" loss, corner를 분리하는 "Push" loss를 다음과 같이 사용한다.

$$L_{pull} = \frac{1}{N}\sum_{k=1}^{N}\left[\left(e_{t_k} - e_k\right)^2 + \left(e_{b_k} - e_k\right)^2\right],$$

$$L_{push} = \frac{1}{N(N-1)}\sum_{k=1}^{N}\sum_{\substack{j=1 \\ j\neq k}}^{N}\max\left(0, \Delta - |e_k - e_j|\right)$$

여기서 e_k는 두 embedding의 평균이며, $\Delta = 1$로 설정했다. 오프셋 loss와 비슷하게, 우리는 ground-truth corner 위치에서만 loss를 적용한다.

3.4. Corner pooling

그림 2와 같이 corner의 존재에 대해 local한 visual evidence가 없는 경우가 많다. 픽셀이 좌측 상단 corner인지 판단하기 위해서는, 객체의 가장 위 경계에 대해서는 오른쪽을 수평으로, 가장 왼쪽 경계에 대해서는 수직으로 아래를 향해야 한다. 따라서 우리는 명시적인 사전 지식을 인코딩하여 corner를 더 잘 localize하기 위해 corner pooling을 제안한다.

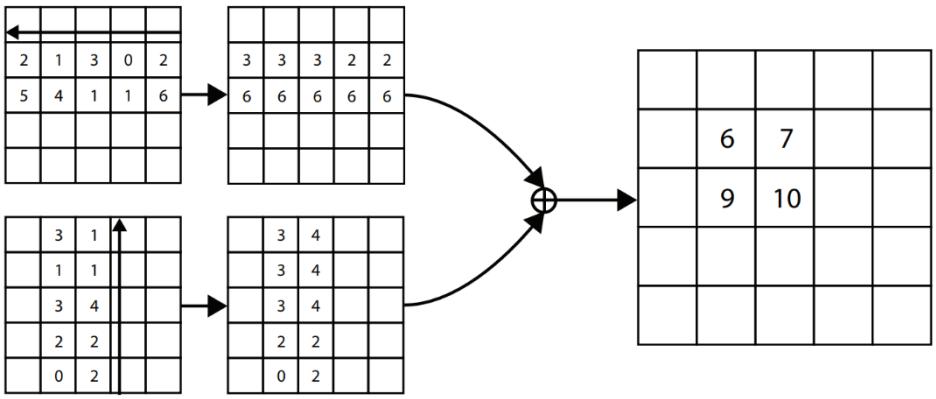

그림 6 좌측 상단 corner pooling layer는 매우 효율적으로 구현될 수 있다. 수평 max-pooling을 위해 왼쪽에서 오른쪽으로, 수직 max-pooling을 위해 아래에서 위로 스캔한다. 그 다음 max-pooled된 feature map 두 개를 더한다.

(i, j) 위치에 있는 픽셀이 좌측 상단 corner인지 여부를 결정하려 한다고 하자. f_t 및 f_l은 좌측 상단 corner pooling layer에 대한 입력인 feature map이 되도록 하고, $f_{t_{ij}}$와 $f_{l_{ij}}$는 각각 f_t와 f_l의 위치의 벡터가 되도록 한다. $H \times W$ feature map에서, corner pooling layer는 먼저 f_t 안의 (i, j)와 (i, H) 사이의 모든 feature vector를 t_{ij}로, f_l 안의 (i, j)와 (W, j) 사이의 feature vector를 l_{ij}로 max-pooling한다. 마지막으로 t_{ij}와 l_{ij}를 더한다. 이 계산은 다음과 같은 방정식으로 표현 가능하다.

$$t_{ij} = \begin{cases} \max\left(f_{t_{ij}}, t_{(i+1)j}\right) & \text{if } i < H \\ f_{t_{Hj}} & \text{otherwise} \end{cases}$$

$$l_{ij} = \begin{cases} \max\left(f_{l_{ij}}, l_{i(j+1)}\right) & \text{if } j < W \\ f_{l_{iW}} & \text{otherwise} \end{cases}$$

여기서 우리는 elementwise max operation을 적용한다. t_{ij}와 l_{ij} 모두 그림 6처럼 동적 프로그래밍을 통해 효율적으로 계산할 수 있다.

우리는 비슷한 방법으로 우측 하단 corner pooling layer를 정의한다. Pooling된 결과를 더하기 전에 $(0, j)$와 (i, j), $(i, 0)$과 (i, j) 사이의 모든 feature vector를 각각 max-pooling한다. Corner pooling layer는 heatmap, embedding, offset을 predict하기 위해 prediction 모듈에 사용된다.

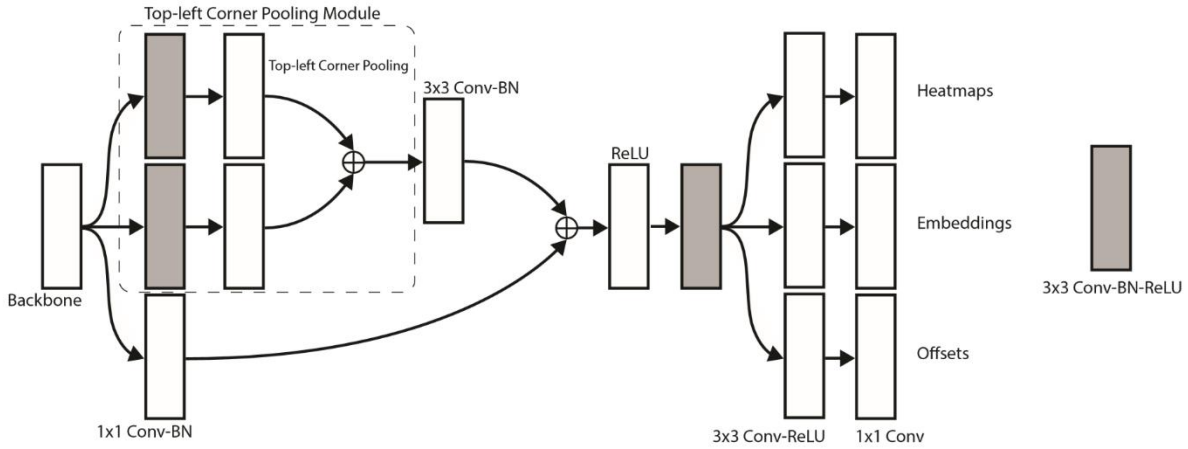

그림 7 prediction 모듈은 수정된 residual block으로 시작하며, 첫 번째 Conv 모듈을 corner pooling 모듈로 교체한다. 수정된 residual block에 이어 Conv 모듈이 이어진다. 우리는 heatmap, embedding, offset을 predict하기 위한 여러 branch를 가지고 있다.

Prediction 모듈의 구조는 그림 7에 나타나 있다. 모듈의 첫 번째 부분은 residual block의 수정 버전이다. 여기서 우리는 처음의 3x3 Conv 모듈을 Corner pooling 모듈로 교체한다. Corner pooling 모듈은 우선 2개의 3x3 128채널 모듈 1에 의해 backbone network의 feature를 처리한 후, corner pooling layer를 적용한다. Residual block 디자인에 따라 pooling된 feature를 256채널 3x3 Conv-BN layer로 공급하고 projection shortcut을 추가한다. 이어 256채널 3x3 Conv 모듈, heatmap, embedding, offset을 생성하기 위한 3개의 Conv-ReLU-Conv layer가 뒤따른다.

3.5. Hourglass network

CornerNet은 hourglass network를 backbone network로 사용한다. 이는 human pose estimation 작업을 위해 처음 도입되었다. 하나 이상의 hourglass 모듈로 구성된 fully CNN이다. 먼저 hourglass 모듈은 일련의 convolution과 max pooling layer로 입력된 feature를 다운샘플링한다. 그 다음 일련의 업샘플링과 convolution layer에 의해 feature를 원래의 해상도로 다시 업샘플링한다. Max pooling layer에서 디테일이 손실되기 때문에, skip layer가 추가되어 디테일을 업샘플링된 feature로 되돌린다. Hourglass 모듈은 단일 통합 구조에서 global 및 local feature를 모두 캡처한다. 여러 hourglass 모듈이 네트워크에 쌓이면 각 모듈이 feature를 재처리하여 high-level 정보를 캡처할 수 있다. 이러한 특성은 hourglass network를 물체 검출에도 이상적인 선택으로 만든다. 실제로 현재 많은 검출기는 이미 hourglass network와 유사한 네트워크를 채택했다.

우리의 hourglass network는 두 개의 모듈로 구성되어 있고, 우리는 모듈의 구조를 약간 수정한다. Max pooling을 사용하는 대신 stride 2만 사용하여 feature 해상도를 줄일 수 있다. 이를 5배 줄이고, 도중에 feature channel 수를 늘린다(256, 384, 384, 512). Feautre를 업샘플링할 때, 우리는 두 개의 residual 모듈을 적용하고 가장 가까운 이웃을 업샘플링한다. 모든 skip connection 또한 2개의 residual 모듈로 구성된다. Hourglass 모듈 중간에 512채널의 4개 residual 모듈이 있다. Hourglass 모듈 전에, 우리는 7x7 convolution 모듈(stride 2, 128채널)을 사용하고, 그 다음 256채널 stride 2 residual block을 사용해서 이미지 해상도를 4배로 줄인다.

학습에서도 중간 supervision을 추가한다. 그러나 우리는 이것이 네트워크의 성능을 떨어트린다는 것을 발견했으므로 네트워크에 중간 prediction을 추가하지 않는다. 우리는 3x3 Conv-BN 모듈을 첫 번째 hourglass 모듈의 입출력 모두에 적용한다. 그다음 element-wise addition에 이어 ReLU와 256채널 residual block을 병합하여 두 번째 hourglass 모듈에 대한 입력으로 사용한다. Hourglass network의 깊이는 104이다. 우리는 전체 네트워크의 마지막 feature만을 사용하여 예측한다.

4. Experiments

4.1. Training Details

We implement CornerNet in PyTorch [29]. The network is randomly initialized under the default setting of PyTorch with no pretraining on any external dataset. As we apply focal loss, we follow [23] to set the biases in the convolution layers that predict the corner heatmaps. During training, we set the input resolution of the network to 511×511, which leads to an output resolution of 128×128. To reduce overfitting, we adopt standard data augmentation techniques including random horizontal flipping, random scaling, random cropping and random color jittering, which includes adjusting the brightness, saturation and contrast of an image. Finally, we apply PCA [20] to the input

image.

We use Adam [18] to optimize the full training loss:

$$\mathrm{L} = \mathrm{L\_det} + \alpha L_{pull} + \beta L_{push} + \gamma L_{off}$$

where α, β and γ are the weights for the pull, push and offset loss respectively. We set both α and β to 0.1 and γ to 1. We find that 1 or larger values of α and β lead to poor performance. We use a batch size of 49 and train the network on 10 Titan X (PASCAL) GPUs (4 images on the master GPU, 5 images per GPU for the rest of the GPUs). To conserve GPU resources, in our ablation experiments, we train the networks for 250k iterations with a learning rate of 2.5×10^{-4}. When we compare our results with other detectors, we train the networks for an extra 250k iterations and reduce the learning rate to 2.5×10^{-5} for the last 50k iterations.

4.2. Testing Details

During testing, we use a simple post-processing algorithm to generate bounding boxes from the heatmaps, embeddings and offsets. We first apply non-maximal suppression (NMS) by using a 3 × 3 max pooling layer on the corner heatmaps. Then we pick the top 100 top-left and top 100 bottom-right corners from the heatmaps. The corner locations are adjusted by the corresponding offsets. We calculate the L1 distances between the embeddings of the top-left and bottomright corners. Pairs that have distances greater than 0.5 or contain corners from different categories are rejected. The average scores of the top-left and bottomright corners are used as the detection scores.

Instead of resizing an image to a fixed size, we maintain the original resolution of the image and pad it with zeros before feeding it to CornerNet. Both the original and flipped images are used for testing. We combine the detections from the original and flipped images, and apply soft-nms [2] to suppress redundant detections. Only the top 100 detections are reported. The average inference time is 244ms per image on a Titan X (PASCAL) GPU.

4.3. MS COCO

We evaluate CornerNet on the very challenging MS COCO dataset [24]. MS COCO contains 80k images for training, 40k for validation and 20k for testing. All images in the training set and 35k images in the validation set are used for training. The remaining 5k images in validation set are used for hyper-parameter searching and ablation study. All results on the test set are submitted to an external server for evaluation. To provide fair comparisons with other detectors, we report our main results on the test-dev set. MS COCO uses average precisions (APs) at different IoUs and APs for different object sizes as the main evaluation metrics.

4.4. Ablation Study

Corner Pooling Corner pooling is a key component of CornerNet. To understand its contribution to performance, we train another network without corner pooling but with the same number of parameters.

Table 1 Ablation on corner pooling on MS COCO validation.

	AP	AP^{50}	AP^{75}	AP^s	AP^m	AP^l
w/o corner pooling	36.5	52.0	38.9	17.6	38.7	48.8
w/ corner pooling	38.5	54.1	41.1	17.7	41.1	52.5
improvement	+2.0	+2.1	+2.2	+0.1	+2.4	+3.7

Tab. 1 shows that adding corner pooling gives significant improvement: 2.0% on AP, 2.1% on AP50 and 2.2% on AP75. We also see that corner pooling is especially helpful for medium and large objects, improving their APs by 2.4% and 3.7% respectivel. This is expected because the topmost, bottommost, leftmost, rightmost boundaries of medium and large objects are likely to be further away from the corner locations.

Reducing penalty to negative locations We reduce the penalty given to negative locations around a positive location, within a radius determined by the size of the object (Sec. 3.2). To understand how this helps train CornerNet, we train one network with no penalty reduction and another network with a fixed radius of 2.5. We compare them with CornerNet on the validation set.

Table 2 Reducing the penalty given to the negative locations near positive locations helps significantly improve the performance of the network

	AP	AP^{50}	AP^{75}	AP^s	AP^m	AP^l
w/o reducing penalty	32.9	49.1	34.8	19.0	37.0	40.7
fixed radius	35.6	52.5	37.7	18.7	38.5	46.0
object-dependent radius	38.5	54.1	41.1	17.7	41.1	52.5

Tab. 2 shows that a fixed radius improves AP over the baseline by 2.7%, APm by 1.5% and APl by 5.3%. Object-dependent radius further improves the AP by 2.9%, APm by 2.6% and APl by 6.5%. In addition, we see that the penalty reduction especially benefits medium and large objects.

Error Analysis CornerNet simultaneously outputs heatmaps, offsets, and embeddings, all of which affect detection performance. An object will be missed if either corner is missed; precise offsets are needed to generate tight bounding boxes; incorrect embeddings will result in many false bounding

boxes. To understand how each part contributes to the final error, we perform an error analysis by replacing the predicted heatmaps and offsets with the ground-truth values and evaluting performance on the validation set.

Table 3 Error analysis. We replace the predicted heatmaps and offsets with the ground-truth values. Using the ground-truth heatmaps alone improves the AP from 38.5% to 74.0%, suggesting that the main bottleneck of CornerNet is detecting corners.

	AP	AP^{50}	AP^{75}	AP^s	AP^m	AP^l
	38.5	54.1	41.1	17.7	41.1	52.5
w/ gt heatmaps	74.0	88.5	79.3	60.8	82.0	82.6
w/ gt heatmaps + offsets	87.1	90.0	86.7	85.0	87.9	83.1

Tab. 3 shows that using the ground-truth corner heatmaps alone improves the AP from 38.5% to 74.0%. APs, APm and APl also increase by 43.1%, 40.9% and 30.1% respectively. If we replace the predicted offsets with the ground-truth offsets, the AP further increases by 13.1% to 87.1%. This suggests that although there is still ample room for improvement in both detecting and grouping corners, the main bottleneck is detecting corners. Fig. 8 shows two qualitative examples of the predicted corners.

Figure 8 Example bounding box predictions overlaid on predicted heatmaps of corners.

4.5. Comparisons with state-of-the-art detectors

We compare CornerNet with other state-of-the-art detectors on MS COCO testdev (Tab. 4). With multi-scale evaluation, CornerNet achieves an AP of 42.1%, the state of the art among existing one-stage methods and competitive with two-stage methods.

Table 4 CornerNet versus others on MS COCO test-dev. CornerNet outperforms all one-stage detectors and achieves results competitive to two-stage detectors

Method	Backbone	AP	AP^{50}	AP^{75}	AP^s	AP^m	AP^l	AR^1	AR^{10}	AR^{100}	AR^s	AR^m	AR^l
Two-stage detectors													
DeNet [39]	ResNet-101	33.8	53.4	36.1	12.3	36.1	50.8	29.6	42.6	43.5	19.2	46.9	64.3
CoupleNet [46]	ResNet-101	34.4	54.8	37.2	13.4	38.1	50.8	30.0	45.0	46.4	20.7	53.1	68.5
Faster R-CNN by G-RMI [16]	Inception-ResNet-v2 [38]	34.7	55.5	36.7	13.5	38.1	52.0	-	-	-	-	-	-
Faster R-CNN+++ [15]	ResNet-101	34.9	55.7	37.4	15.6	38.7	50.9	-	-	-	-	-	-
Faster R-CNN w/ FPN [22]	ResNet-101	36.2	59.1	39.0	18.2	39.0	48.2	-	-	-	-	-	-
Faster R-CNN w/ TDM [35]	Inception-ResNet-v2	36.8	57.7	39.2	16.2	39.8	52.1	31.6	49.3	51.9	28.1	56.6	71.1
D-FCN [7]	Aligned-Inception-ResNet	37.5	58.0	-	19.4	40.1	52.5	-	-	-	-	-	-
Regionlets [43]	ResNet-101	39.3	59.8	-	21.7	43.7	50.9	-	-	-	-	-	-
Mask R-CNN [13]	ResNeXt-101	39.8	62.3	43.4	22.1	43.2	51.2	-	-	-	-	-	-
Soft-NMS [2]	Aligned-Inception-ResNet	40.9	62.8	-	23.3	43.6	53.3	-	-	-	-	-	-
LH R-CNN [21]	ResNet-101	41.5	-	-	25.2	45.3	53.1	-	-	-	-	-	-
Fitness-NMS [40]	ResNet-101	41.8	60.9	44.9	21.5	45.0	57.5	-	-	-	-	-	-
Cascade R-CNN [4]	ResNet-101	42.8	62.1	46.3	23.7	45.5	55.2	-	-	-	-	-	-
D-RFCN + SNIP [37]	DPN-98 [5]	45.7	67.3	51.1	29.3	48.8	57.1	-	-	-	-	-	-
One-stage detectors													
YOLOv2 [31]	DarkNet-19	21.6	44.0	19.2	5.0	22.4	35.5	20.7	31.6	33.3	9.8	36.5	54.4
DSOD300 [33]	DS/64-192-48-1	29.3	47.3	30.6	9.4	31.5	47.0	27.3	40.7	43.0	16.7	47.1	65.0
GRP-DSOD320 [34]	DS/64-192-48-1	30.0	47.9	31.8	10.9	33.6	46.3	28.0	42.1	44.5	18.8	49.1	65.0
SSD513 [25]	ResNet-101	31.2	50.4	33.3	10.2	34.5	49.8	28.3	42.1	44.4	17.6	49.2	65.8
DSSD513 [10]	ResNet-101	33.2	53.3	35.2	13.0	35.4	51.1	28.9	43.5	46.2	21.8	49.1	66.4
RefineDet512 (single scale) [45]	ResNet-101	36.4	57.5	39.5	16.6	39.9	51.4	-	-	-	-	-	-
RetinaNet800 [23]	ResNet-101	39.1	59.1	42.3	21.8	42.7	50.2	-	-	-	-	-	-
RefineDet512 (multi scale) [45]	ResNet-101	41.8	62.9	45.7	25.6	45.1	54.1	-	-	-	-	-	-
CornerNet511 (single scale)	Hourglass-104	40.5	56.5	43.1	19.4	42.7	53.9	35.3	54.3	59.1	37.4	61.9	76.9
CornerNet511 (multi scale)	Hourglass-104	42.1	57.8	45.3	20.8	44.8	56.7	36.4	55.7	60.0	38.5	62.7	77.4

5. Conclusion

We have presented CornerNet, a new approach to object detection that detects bounding boxes as pairs of corners. We evaluate CornerNet on MS COCO and demonstrate competitive results.