CornerNet: Detecting Objects as Paired Keypoints

Hei Law(✉) and Jia Deng

University of Michigan, Ann Arbor, USA
{heilaw,jiadeng}@umich.edu

Abstract. We propose CornerNet, a new approach to object detection where we detect an object bounding box as a pair of keypoints, the top-left corner and the bottom-right corner, using a single convolution neural network. By detecting objects as paired keypoints, we eliminate the need for designing a set of anchor boxes commonly used in prior single-stage detectors. In addition to our novel formulation, we introduce corner pooling, a new type of pooling layer that helps the network better localize corners. Experiments show that CornerNet achieves a 42.1% AP on MS COCO, outperforming all existing one-stage detectors.

Keyword: Object detection

1 Introduction

Object detectors based on convolutional neural networks (ConvNets) [15,20,36] have achieved state-of-the-art results on various challenging benchmarks [8,9,24]. A common component of state-of-the-art approaches is anchor boxes [25,32], which are boxes of various sizes and aspect ratios that serve as detection candidates. Anchor boxes are extensively used in one-stage detectors [10,23,25,31], which can achieve results highly competitive with two-stage detectors [11–13,32] while being more efficient. One-stage detectors place anchor boxes densely over an image and generate final box predictions by scoring anchor boxes and refining their coordinates through regression.

But the use of anchor boxes has two drawbacks. First, we typically need a very large set of anchor boxes, e.g. more than 40k in DSSD [10] and more than 100k in RetinaNet [23]. This is because the detector is trained to classify whether each anchor box sufficiently overlaps with a ground truth box, and a large number of anchor boxes is needed to ensure sufficient overlap with most ground truth boxes. As a result, only a tiny fraction of anchor boxes will overlap with ground truth; this creates a huge imbalance between positive and negative anchor boxes and slows down training [23].

Second, the use of anchor boxes introduces many hyperparameters and design choices. These include how many boxes, what sizes, and what aspect ratios. Such choices have largely been made via ad-hoc heuristics, and can become even more

V. Ferrari et al. (Eds.): ECCV 2018, LNCS 11218, pp. 765–781, 2018.
https://doi.org/10.1007/978-3-030-01264-9_45

complicated when combined with multiscale architectures where a single network makes separate predictions at multiple resolutions, with each scale using different features and its own set of anchor boxes [10,23,25].

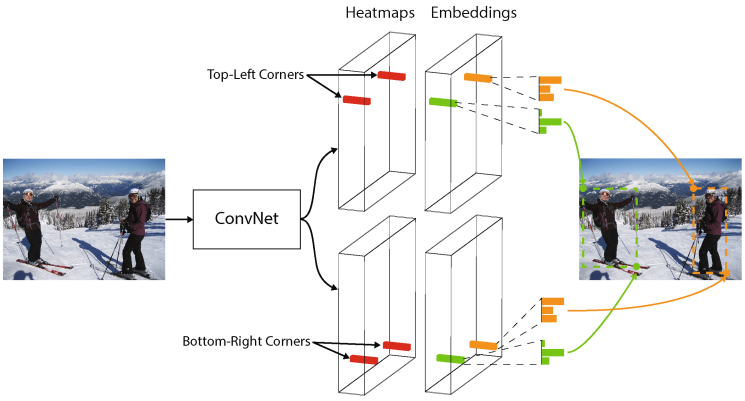

Fig. 1. We detect an object as a pair of bounding box corners grouped together. A convolutional network outputs a heatmap for all top-left corners, a heatmap for all bottom-right corners, and an embedding vector for each detected corner. The network is trained to predict similar embeddings for corners that belong to the same object.

In this paper we introduce CornerNet, a new one-stage approach to object detection that does away with anchor boxes. We detect an object as a pair of keypoints—the top-left corner and bottom-right corner of the bounding box. We use a single convolutional network to predict a heatmap for the top-left corners of all instances of the same object category, a heatmap for all bottom-right corners, and an embedding vector for each detected corner. The embeddings serve to group a pair of corners that belong to the same object—the network is trained to predict similar embeddings for them. Our approach greatly simplifies the output of the network and eliminates the need for designing anchor boxes. Our approach is inspired by the associative embedding method proposed by Newell et al. [27], who detect and group keypoints in the context of multiperson human-pose estimation. Figure 1 illustrates the overall pipeline of our approach.

Fig. 2. Often there is no local evidence to determine the location of a bounding box corner. We address this issue by proposing a new type of pooling layer.

Another novel component of CornerNet is *corner pooling*, a new type of pooling layer that helps a convolutional network better localize corners of bounding boxes. A corner of a bounding box is often outside the object—consider the case of a circle as well as the examples in Fig. 2. In such cases a corner cannot be localized based on local evidence. Instead, to determine whether there is a top-left corner at a pixel location, we need to look horizontally towards the right for the topmost boundary of the object, and look vertically towards the bottom for the leftmost boundary. This motivates our corner pooling layer: it takes in two feature maps; at each pixel location it max-pools all feature vectors to the right from the first feature map, max-pools all feature vectors directly below from the second feature map, and then adds the two pooled results together. An example is shown in Fig. 3.

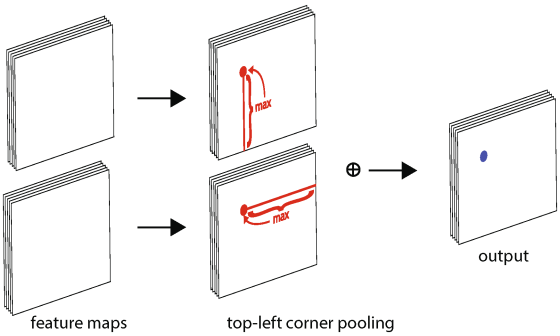

Fig. 3. Corner pooling: for each channel, we take the maximum values *(red dots)* in two directions *(red lines)*, each from a separate feature map, and add the two maximums together *(blue dot)*. (Color figure online)

We hypothesize two reasons why detecting corners would work better than bounding box centers or proposals. First, the center of a box can be harder to localize because it depends on all 4 sides of the object, whereas locating a corner depends on 2 sides and is thus easier, and even more so with corner pooling, which encodes some explicit prior knowledge about the definition of corners. Second, corners provide a more efficient way of densely discretizing the space of boxes: we just need $O(wh)$ corners to represent $O(w^2h^2)$ possible anchor boxes.

We demonstrate the effectiveness of CornerNet on MS COCO [24]. CornerNet achieves a 42.1% AP, outperforming all existing one-stage detectors. In addition, through ablation studies we show that corner pooling is critical to the superior performance of CornerNet. Code is available at https://github.com/umich-vl/CornerNet.

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 [1,3,22,35], 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 top-left, 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

In CornerNet, we detect an object as a pair of keypoints—the top-left corner and bottom-right corner of the bounding box. A convolutional network predicts two sets of heatmaps to represent the locations of corners of different object categories, one set for the top-left corners and the other for the bottom-right corners. The network also predicts an embedding vector for each detected corner [27] such that the distance between the embeddings of two corners from the same object is small. To produce tighter bounding boxes, the network also predicts offsets to slightly adjust the locations of the corners. With the predicted heatmaps, embeddings and offsets, we apply a simple post-processing algorithm to obtain the final bounding boxes.

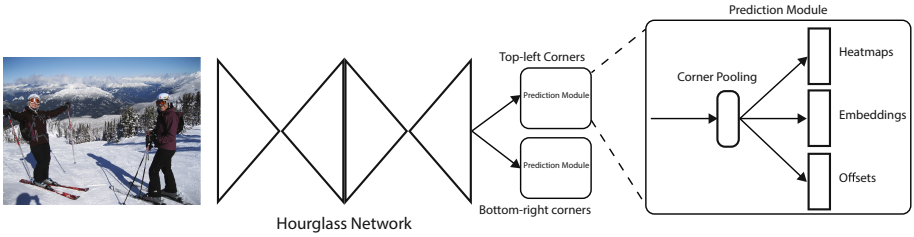

Fig. 4. Overview of CornerNet. The backbone network is followed by two prediction modules, one for the top-left corners and the other for the bottom-right corners. Using the predictions from both modules, we locate and group the corners.

Figure 4 provides an overview of CornerNet. We use the hourglass network [28] as the backbone network of CornerNet. The hourglass network is followed by two prediction modules. One module is for the top-left corners, while the other one is for the bottom-right corners. Each module has its own corner pooling module to pool features from the hourglass network before predicting the heatmaps, embeddings and offsets. Unlike many other object detectors, we do not use features from different scales to detect objects of different sizes. We only apply both modules to the output of the hourglass network.

3.2 Detecting Corners

We predict two sets of heatmaps, one for top-left corners and one for bottom-right corners. Each set of heatmaps has C channels, where C is the number of categories, and is of size $H \times W$. There is no background channel. Each channel is a binary mask indicating the locations of the corners for a class.

Fig. 5. "Ground-truth" heatmaps for training. Boxes *(green dotted rectangles)* whose corners are within the radii of the positive locations *(orange circles)* still have large overlaps with the ground-truth annotations *(red solid rectangles)*. (Color figure online)

For each corner, there is one ground-truth positive location, and all other locations are negative. During training, instead of equally penalizing negative locations, we reduce the penalty given to negative locations within a radius of the positive location. This is because a pair of false corner detections, if they are close to their respective ground truth locations, can still produce a box that sufficiently overlaps the ground-truth box (Fig. 5). We determine the radius by the size of an object by ensuring that a pair of points within the radius would generate a bounding box with at least t IoU with the ground-truth annotation (we set t to 0.7 in all experiments). Given the radius, the amount of penalty reduction is given by an unnormalized 2D Gaussian, $e^{-\frac{x^2+y^2}{2\sigma^2}}$, whose center is at the positive location and whose σ is 1/3 of the radius.

Let p_{cij} be the score at location (i, j) for class c in the predicted heatmaps, and let y_{cij} be the "ground-truth" heatmap augmented with the unnormalized Gaussians. We design a variant of focal loss [23]:

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

where N is the number of objects in an image, and α and β are the hyper-parameters which control the contribution of each point (we set α to 2 and β to 4 in all experiments). With the Gaussian bumps encoded in y_{cij}, the $(1 - y_{cij})$ term reduces the penalty around the ground truth locations.

Many networks [15,28] involve downsampling layers to gather global information and to reduce memory usage. When they are applied to an image fully convolutionally, the size of the output is usually smaller than the image. Hence, a location (x, y) in the image is mapped to the location $\left(\lfloor \frac{x}{n} \rfloor, \lfloor \frac{y}{n} \rfloor\right)$ in the heatmaps, where n is the downsampling factor. When we remap the locations from the heatmaps to the input image, some precision may be lost, which can greatly affect the IoU of small bounding boxes with their ground truths. To address this issue we predict location offsets to slightly adjust the corner locations before remapping them to the input resolution.

$$\boldsymbol{o}_k = \left(\frac{x_k}{n} - \left\lfloor \frac{x_k}{n} \right\rfloor, \frac{y_k}{n} - \left\lfloor \frac{y_k}{n} \right\rfloor \right) \tag{2}$$

where $\boldsymbol{o}_k$ is the offset, x_k and y_k are the x and y coordinate for corner k. In particular, we predict one set of offsets shared by the top-left corners of all categories, and another set shared by the bottom-right corners. For training, we apply the smooth L1 Loss [11] at ground-truth corner locations:

$$L_{off} = \frac{1}{N} \sum_{k=1}^{N} \text{SmoothL1Loss} \left(\boldsymbol{o}_k, \hat{\boldsymbol{o}}_k \right) \tag{3}$$

3.3 Grouping Corners

Multiple objects may appear in an image, and thus multiple top-left and bottom-right corners may be detected. We need to determine if a pair of the top-left corner and bottom-right corner is from the same bounding box. Our approach is inspired by the Associative Embedding method proposed by Newell et al. [27] for the task of multi-person pose estimation. Newell et al. detect all human joints and generate an embedding for each detected joint. They group the joints based on the distances between the embeddings.

The idea of associative embedding is also applicable to our task. The network predicts an embedding vector for each detected corner such that if a top-left corner and a bottom-right corner belong to the same bounding box, the distance between their embeddings should be small. We can then group the corners based

on the distances between the embeddings of the top-left and bottom-right corners. The actual values of the embeddings are unimportant. Only the distances between the embeddings are used to group the corners.

We follow Newell et al. [27] and use embeddings of 1 dimension. Let e_{t_k} be the embedding for the top-left corner of object k and e_{b_k} for the bottom-right corner. As in [26], we use the "pull" loss to train the network to group the corners and the "push" loss to separate the corners:

$$L_{pull} = \frac{1}{N}\sum_{k=1}^{N}\left[\left(e_{t_k} - e_k\right)^2 + \left(e_{b_k} - e_k\right)^2\right], \tag{4}$$

$$L_{push} = \frac{1}{N(N-1)}\sum_{k=1}^{N}\sum_{\substack{j=1\\ j\neq k}}^{N}\max\left(0, \Delta - |e_k - e_j|\right), \tag{5}$$

where e_k is the average of e_{t_k} and e_{b_k} and we set Δ to be 1 in all our experiments. Similar to the offset loss, we only apply the losses at the ground-truth corner location.

3.4 Corner Pooling

As shown in Fig. 2, there is often no local visual evidence for the presence of corners. To determine if a pixel is a top-left corner, we need to look horizontally towards the right for the topmost boundary of an object and vertically towards the bottom for the leftmost boundary. We thus propose *corner pooling* to better localize the corners by encoding explicit prior knowledge.

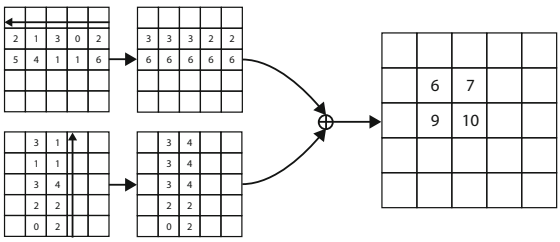

Fig. 6. The top-left corner pooling layer can be implemented very efficiently. We scan from left to right for the horizontal max-pooling and from bottom to top for the vertical max-pooling. We then add two max-pooled feature maps.

Suppose we want to determine if a pixel at location (i, j) is a top-left corner. Let f_t and f_l be the feature maps that are the inputs to the top-left corner pooling layer, and let $f_{t_{ij}}$ and $f_{l_{ij}}$ be the vectors at location (i, j) in f_t and f_l respectively. With $H \times W$ feature maps, the corner pooling layer first max-pools all feature vectors between (i, j) and (i, H) in f_t to a feature vector t_{ij}, and max-pools all feature vectors between (i, j) and (W, j) in f_l to a feature vector

l_{ij}. Finally, it adds t_{ij} and l_{ij} together. This computation can be expressed by the following equations:

$$t_{ij} = \begin{cases} \max\left(f_{t_{ij}}, t_{(i+1)j}\right) & \text{if } i < H \\ f_{t_{Hj}} & \text{otherwise} \end{cases} \tag{6}$$

$$l_{ij} = \begin{cases} \max\left(f_{l_{ij}}, l_{i(j+1)}\right) & \text{if } j < W \\ f_{l_{iW}} & \text{otherwise} \end{cases} \tag{7}$$

where we apply an elementwise max operation. Both t_{ij} and l_{ij} can be computed efficiently by dynamic programming as shown Fig. 6.

We define bottom-right corner pooling layer in a similar way. It max-pools all feature vectors between $(0, j)$ and (i, j), and all feature vectors between $(i, 0)$ and (i, j) before adding the pooled results. The corner pooling layers are used in the prediction modules to predict heatmaps, embeddings and offsets.

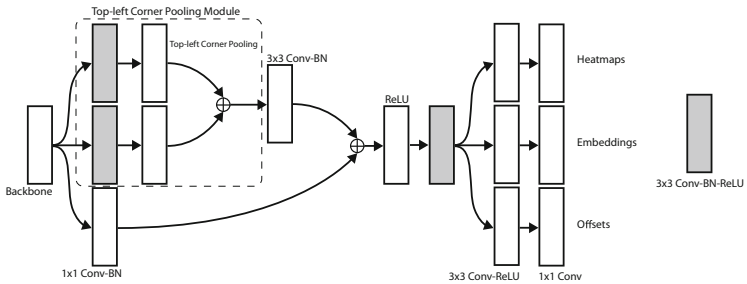

Fig. 7. The prediction module starts with a modified residual block, in which we replace the first convolution module with our corner pooling module. The modified residual block is then followed by a convolution module. We have multiple branches for predicting the heatmaps, embeddings and offsets.

The architecture of the prediction module is shown in Fig. 7. The first part of the module is a modified version of the residual block [15]. In this modified residual block, we replace the first 3×3 convolution module with a corner pooling module, which first processes the features from the backbone network by two 3×3 convolution modules[1] with 128 channels and then applies a corner pooling layer. Following the design of a residual block, we then feed the pooled features into a 3×3 Conv-BN layer with 256 channels and add back the projection shortcut. The modified residual block is followed by a 3×3 convolution module with 256 channels, and 3 Conv-ReLU-Conv layers to produce the heatmaps, embeddings and offsets.

[1] Unless otherwise specified, our convolution module consists of a convolution layer, a BN layer [17] and a ReLU layer.

3.5 Hourglass Network

CornerNet uses the hourglass network [28] as its backbone network. The hourglass network was first introduced for the human pose estimation task. It is a fully convolutional neural network that consists of one or more hourglass modules. An hourglass module first downsamples the input features by a series of convolution and max pooling layers. It then upsamples the features back to the original resolution by a series of upsampling and convolution layers. Since details are lost in the max pooling layers, skip layers are added to bring back the details to the upsampled features. The hourglass module captures both global and local features in a single unified structure. When multiple hourglass modules are stacked in the network, the hourglass modules can reprocess the features to capture higher-level of information. These properties make the hourglass network an ideal choice for object detection as well. In fact, many current detectors [10,19,22,35] already adopted networks similar to the hourglass network.

Our hourglass network consists of two hourglasses, and we make some modifications to the architecture of the hourglass module. Instead of using max pooling, we simply use stride 2 to reduce feature resolution. We reduce feature resolutions 5 times and increase the number of feature channels along the way $(256, 384, 384, 384, 512)$. When we upsample the features, we apply 2 residual modules followed by a nearest neighbor upsampling. Every skip connection also consists of 2 residual modules. There are 4 residual modules with 512 channels in the middle of an hourglass module. Before the hourglass modules, we reduce the image resolution by 4 times using a 7×7 convolution module with stride 2 and 128 channels followed by a residual block [15] with stride 2 and 256 channels.

Following [28], we also add intermediate supervision in training. However, we do not add back the intermediate predictions to the network as we find that this hurts the performance of the network. We apply a 3×3 Conv-BN module to both the input and output of the first hourglass module. We then merge them by element-wise addition followed by a ReLU and a residual block with 256 channels, which is then used as the input to the second hourglass module. The depth of the hourglass network is 104. Unlike many other state-of-the-art detectors, we only use the features from the last layer of the whole network to make predictions.

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:

$$L = L_{det} + \alpha L_{pull} + \beta L_{push} + \gamma L_{off} \tag{8}$$

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 bottom-right 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 bottom-right 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 244 ms 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

Table 1 shows that adding corner pooling gives significant improvement: 2.0% on AP, 2.1% on AP^{50} and 2.2% on AP^{75}. We also see that corner pooling is especially helpful for medium and large objects, improving their APs by 2.4% and 3.7% respectively. 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 (Sect. 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

Table 2 shows that a fixed radius improves AP over the baseline by 2.7%, AP^m by 1.5% and AP^l by 5.3%. Object-dependent radius further improves the AP by 2.9%, AP^m by 2.6% and AP^l 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

Table 3 shows that using the ground-truth corner heatmaps alone improves the AP from 38.5% to 74.0%. AP^s, AP^m and AP^l 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. Figure 8 shows two qualitative examples of the predicted corners.

Fig. 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 test-dev (Table 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.

Acknowledgements. Toyota Research Institute ("TRI") provided funds to assist the authors with their research but this article solely reflects the opinions and conclusions of its authors and not TRI or any other Toyota entity.

References

1. Bell, S., Lawrence Zitnick, C., Bala, K., Girshick, R.: Inside-outside net: detecting objects in context with skip pooling and recurrent neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2874–2883 (2016)
2. Bodla, N., Singh, B., Chellappa, R., Davis, L.S.: Soft-NMS improving object detection with one line of code. In: 2017 IEEE International Conference on Computer Vision (ICCV), pp. 5562–5570. IEEE (2017)
3. Cai, Z., Fan, Q., Feris, R.S., Vasconcelos, N.: A unified multi-scale deep convolutional neural network for fast object detection. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9908, pp. 354–370. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46493-0_22
4. Cai, Z., Vasconcelos, N.: Cascade R-CNN: delving into high quality object detection. arXiv preprint arXiv:1712.00726 (2017)

5. Chen, Y., Li, J., Xiao, H., Jin, X., Yan, S., Feng, J.: Dual path networks. In: Advances in Neural Information Processing Systems, pp. 4470–4478 (2017)
6. Dai, J., Li, Y., He, K., Sun, J.: R-FCN: object detection via region-based fully convolutional networks. arXiv preprint arXiv:1605.06409 (2016)
7. Dai, J., Qi, H., Xiong, Y., Li, Y., Zhang, G., Hu, H., Wei, Y.: Deformable convolutional networks. CoRR, abs/1703.06211, vol. 1(2), p. 3 (2017)
8. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: a large-scale hierarchical image database. In: IEEE Conference on Computer Vision and Pattern Recognition CVPR 2009, pp. 248–255. IEEE (2009)
9. Everingham, M., Eslami, S.A., Van Gool, L., Williams, C.K., Winn, J., Zisserman, A.: The pascal visual object classes challenge: a retrospective. Int. J. Comput. Vis. **111**(1), 98–136 (2015)
10. Fu, C.Y., Liu, W., Ranga, A., Tyagi, A., Berg, A.C.: DSSD: Deconvolutional single shot detector. arXiv preprint arXiv:1701.06659 (2017)
11. Girshick, R.: FAST R-CNN. arXiv preprint arXiv:1504.08083 (2015)
12. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 580–587 (2014)
13. He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask R-CNN. arxiv preprint arxiv: 170306870 (2017)
14. He, K., Zhang, X., Ren, S., Sun, J.: Spatial pyramid pooling in deep convolutional networks for visual recognition. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV 2014. LNCS, vol. 8691, pp. 346–361. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10578-9_23
15. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
16. Huang, J., et al.: Speed/accuracy trade-offs for modern convolutional object detectors. In: IEEE CVPR (2017)
17. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: International Conference on Machine Learning, pp. 448–456 (2015)
18. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
19. Kong, T., Sun, F., Yao, A., Liu, H., Lu, M., Chen, Y.: Ron: reverse connection with objectness prior networks for object detection. arXiv preprint arXiv:1707.01691 (2017)
20. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
21. Li, Z., Peng, C., Yu, G., Zhang, X., Deng, Y., Sun, J.: Light-head R-CNN: in defense of two-stage object detector. arXiv preprint arXiv:1711.07264 (2017)
22. Lin, T.Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S.: Feature pyramid networks for object detection. arXiv preprint arXiv:1612.03144 (2016)
23. Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollár, P.: Focal loss for dense object detection. arXiv preprint arXiv:1708.02002 (2017)
24. Lin, T.-Y., et al.: Microsoft COCO: common objects in context. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV 2014. LNCS, vol. 8693, pp. 740–755. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10602-1_48

25. Liu, W., et al.: SSD: single shot multibox detector. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9905, pp. 21–37. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46448-0_2
26. Newell, A., Deng, J.: Pixels to graphs by associative embedding. In: Advances in Neural Information Processing Systems, pp. 2168–2177 (2017)
27. Newell, A., Huang, Z., Deng, J.: Associative embedding: end-to-end learning for joint detection and grouping. In: Advances in Neural Information Processing Systems, pp. 2274–2284 (2017)
28. Newell, A., Yang, K., Deng, J.: Stacked hourglass networks for human pose estimation. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9912, pp. 483–499. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46484-8_29
29. Paszke, A., et al.: Automatic differentiation in pytorch (2017)
30. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: unified, real-time object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 779–788 (2016)
31. Redmon, J., Farhadi, A.: Yolo9000: better, faster, stronger. arXiv preprint 1612 (2016)
32. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: towards real-time object detection with region proposal networks. In: Advances in Neural Information Processing Systems, pp. 91–99 (2015)
33. Shen, Z., Liu, Z., Li, J., Jiang, Y.G., Chen, Y., Xue, X.: DSOD: learning deeply supervised object detectors from scratch. In: The IEEE International Conference on Computer Vision (ICCV), vol. 3, p. 7 (2017)
34. Shen, Z., et al.: Learning object detectors from scratch with gated recurrent feature pyramids. arXiv preprint arXiv:1712.00886 (2017)
35. Shrivastava, A., Sukthankar, R., Malik, J., Gupta, A.: Beyond skip connections: top-down modulation for object detection. arXiv preprint arXiv:1612.06851 (2016)
36. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
37. Singh, B., Davis, L.S.: An analysis of scale invariance in object detection-snip. arXiv preprint arXiv:1711.08189 (2017)
38. Szegedy, C., Ioffe, S., Vanhoucke, V., Alemi, A.A.: Inception-v4, inception-resnet and the impact of residual connections on learning. In: AAAI, vol. 4, p. 12 (2017)
39. Tychsen-Smith, L., Petersson, L.: Denet: scalable real-time object detection with directed sparse sampling. arXiv preprint arXiv:1703.10295 (2017)
40. Tychsen-Smith, L., Petersson, L.: Improving object localization with fitness nms and bounded iou loss. arXiv preprint arXiv:1711.00164 (2017)
41. Uijlings, J.R., van de Sande, K.E., Gevers, T., Smeulders, A.W.: Selective search for object recognition. Int. J. Comput. Vis. **104**(2), 154–171 (2013)
42. Xiang, Y., Choi, W., Lin, Y., Savarese, S.: Subcategory-aware convolutional neural networks for object proposals and detection. arXiv preprint arXiv:1604.04693 (2016)
43. Xu, H., Lv, X., Wang, X., Ren, Z., Chellappa, R.: Deep regionlets for object detection. arXiv preprint arXiv:1712.02408 (2017)
44. Zhai, Y., Fu, J., Lu, Y., Li, H.: Feature selective networks for object detection. arXiv preprint arXiv:1711.08879 (2017)
45. Zhang, S., Wen, L., Bian, X., Lei, Z., Li, S.Z.: Single-shot refinement neural network for object detection. arXiv preprint arXiv:1711.06897 (2017)

46. Zhu, Y., Zhao, C., Wang, J., Zhao, X., Wu, Y., Lu, H.: Couplenet: coupling global structure with local parts for object detection. In: Proceedings of International Conference on Computer Vision (ICCV) (2017)
47. Zitnick, C.L., Dollár, P.: Edge boxes: locating object proposals from edges. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV 2014. LNCS, vol. 8693, pp. 391–405. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10602-1_26