CornerNet-Lite: Efficient Keypoint Based Object Detection

Hei Law Yun Teng Olga Russakovsky Jia Deng
Princeton University

Abstract

Keypoint-based methods are a relatively new paradigm in object detection, eliminating the need for anchor boxes and offering a simplified detection framework. Keypoint-based CornerNet achieves state of the art accuracy among single-stage detectors. However, this accuracy comes at high processing cost. In this work, we tackle the problem of efficient keypoint-based object detection and introduce CornerNet-Lite. CornerNet-Lite is a combination of two efficient variants of CornerNet: CornerNet-Saccade, which uses an attention mechanism to eliminate the need for exhaustively processing all pixels of the image, and CornerNet-Squeeze, which introduces a new compact backbone architecture. Together these two variants address the two critical use cases in efficient object detection: improving efficiency without sacrificing accuracy, and improving accuracy at real-time efficiency. CornerNet-Saccade is suitable for offline processing, improving the efficiency of CornerNet by 6.0x and the AP by 1.0% on COCO. CornerNet-Squeeze is suitable for real-time detection, improving both the efficiency and accuracy of the popular real-time detector YOLOv3 (34.4% AP at 34ms for CornerNet-Squeeze compared to 33.0% AP at 39ms for YOLOv3 on COCO). Together these contributions for the first time reveal the potential of keypoint-based detection to be useful for applications requiring processing efficiency.

1. Introduction

Keypoint-based object detection [53, 56, 26] is a class of methods that generate object bounding boxes by detecting and grouping their keypoints. CornerNet [26], the state-of-the-art among them, detects and groups the top-left and bottom-right corners of bounding boxes; it uses a stacked hourglass network [39] to predict the heatmaps of the corners and then uses associate embeddings [38] to group them. CornerNet allows a simplified design that eliminates the need for anchor boxes [46], and has achieved state-of-the-art accuracy on COCO [32] among single-stage detectors.

However, a major drawback of CornerNet is its inference speed. It achieves an average precision (AP) of 42.2% on COCO at an inference cost of 1.147s per image, which is too slow for video applications that require real-time or interactive rates. Although one can easily speed up inference by reducing the number of pixels processed (e.g. by reducing the number of scales of processing or the image resolution), this can cause a large accuracy drop. For example, single-scale processing combined with reducing the input resolution can speed up the inference of CornerNet to 42ms per image, comparable to the 39ms of the popular fast detector YOLOv3 [45], but would decrease the AP to 25.6% which is much lower than YOLOv3's 33.0%. This makes CornerNet less competitive with alternatives in terms of the accuracy-efficiency tradeoff.

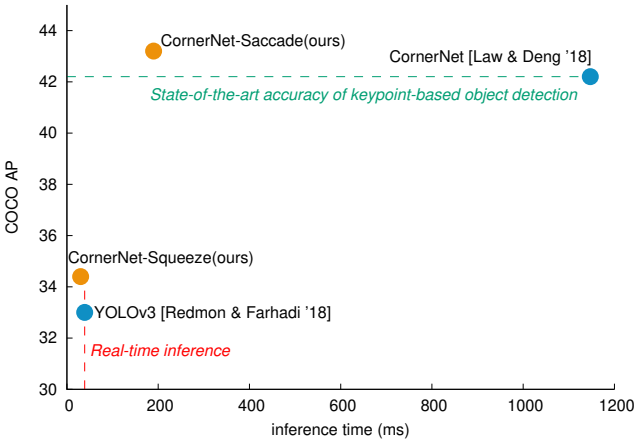

Figure 1: We introduce CornerNet-Saccade and CornerNet-Squeeze (collectively as CornerNet-Lite), two efficient object detectors based on CornerNet [26], a state-of-the-art keypoint based object detector. CornerNet-Saccade speeds up the original CornerNet by 6.0x with a 1% increase in AP. CornerNet-Squeeze is faster and more accurate than YOLOv3 [45], the state-of-the-art real time detector. All detectors are tested on the same machine with a 1080Ti GPU and an Intel Core i7-7700k CPU.

In this paper we seek to improve the inference efficiency of CornerNet. The efficiency of any object detector can be improved along two orthogonal directions: reducing the number of pixels processed and reducing the amount of pro-

cessing per pixel. We explore both directions and introduce two efficient variants of CornerNet: *CornerNet-Saccade* and *CornerNet-Squeeze*, which we refer to collectively as *CornerNet-Lite*.

CornerNet-Saccade speeds up inference by reducing the number of pixels to process. It uses an attention mechanism similar to saccades in human vision [58, 1]. It starts with a downsized full image and generates an attention map, which is then zoomed in on and processed further by the model. This differs from the original CornerNet in that it is applied fully convolutionally across multiple scales. By selecting a subset of crops to examine in high resolution, CornerNet-Saccade improves speed while improving the accuracy. Experiments on COCO show that CornerNet-Saccade achieves an AP of 43.2% at 190ms per image, a 1% increase in AP and a 6.0x speed-up over the original CornerNet.

CornerNet-Squeeze speeds up inference by reducing the amount of processing per pixel. It incorporates ideas from SqueezeNet [19] and MobileNets [15], and introduces a new, compact hourglass backbone that makes extensive use of 1×1 convolution, bottleneck layer, and depth-wise separable convolution. With the new hourglass backbone, CornerNet-Squeeze achieves an AP of 34.4% on COCO at 30ms, simultaneously more accurate and faster than YOLOv3 (33.0% at 39ms).

A natural question is whether CornerNet-Squeeze can be combined with saccades to improve its efficiency even further. Somewhat surprisingly, our experiments give a negative answer: *CornerNet-Squeeze-Saccade* turns out slower and less accurate than CornerNet-Squeeze. This is because for saccades to help, the network needs to be able to generate sufficiently accurate attention maps, but the ultra-compact architecture of CornerNet-Squeeze does not have this extra capacity. In addition, the original CornerNet is applied at multiple scales, which provides ample room for saccades to cut down on the number of pixels to process. In contrast, CornerNet-Squeeze is already applied at a single scale due to the ultra-tight inference budget, which provides much less room for saccades to save.

Significance and novelty: Collectively, these two variants of CornerNet-Lite make the keypoint-based approach competitive, covering two popular use cases: CornerNet-Saccade for offline processing, improving efficiency without sacrificing accuracy, and CornerNet-Squeeze for real-time processing, improving accuracy without sacrificing efficiency.

Both variants of CornerNet-Lite are technically novel. CornerNet-Saccade is the first to integrate saccades with keypoint-based object detection. Its key difference from prior work lies in how each crop (of pixels or feature maps) is processed. Prior work that employs saccade-like mechanisms either detects a single object per crop (e.g. Faster R-CNN [46]) or produces multiple detections per crop with a two-stage network involving additional sub-crops (e.g. AutoFocus [37]). In contrast, CornerNet-Saccade produces multiple detections per crop with a single-stage network.

CornerNet-Squeeze is the first to integrate SqueezeNet with the stacked hourglass architecture and to apply such a combination on object detection. Prior works that employ the hourglass architecture have excelled at achieving competitive accuracy, but it was unclear whether and how the hourglass architecture can be competitive in terms of efficiency. Our design and results show that this is possible for the first time, particularly in the context of object detection.

Contributions Our contributions are three-fold: (1) We propose CornerNet-Saccade and CornerNet-Squeeze, two novel approaches to improving the efficiency of keypoint-based object detection; (2) On COCO, we improve the efficiency of state-of-the-art keypoint based detection by 6 fold and the AP from 42.2% to 43.2%, (3) On COCO, we improve both the accuracy and efficiency of state-of-the art real-time object detection (to 34.4% at 30ms from 33.0% at 39ms of YOLOv3).

Code is available at https://github.com/princeton-vl/CornerNet-Lite.

2. Related Work

Saccades in Object Detection. Saccades in human vision refers to a sequence of rapid eye movements to fixate different image regions. In the context of object detection algorithms, we use the term broadly to mean selectively cropping and processing image regions (sequentially or in parallel, pixels or features) during inference.

There has been a long history of using saccades in object detection to speed up inference. For example, a special case of saccades is a cascade that repeatedly selects a subset of regions for further processing, as exemplified by the Viola-Jones face detector [54]. The idea of saccades has taken diverse forms in various approaches, but can be roughly categorized by how each crop is processed, in particular, what kind of output is produced after processing each crop.

Saccades in R-CNN [11], Fast R-CNN [10], and Faster R-CNN [46] take the form of crops representing potential objects. After processing, each crop is either rejected or converted to a single labeled box through classification and regression. Cascade R-CNN [4] extends Faster R-CNN by using a cascade of classifiers and regressors to iteratively reject or refine each proposal. The saccades in all these R-CNN variants are thus *single-type and single-object*, in that there is a single type of processing of crops, and each crop produces at most a single object.

AutoFocus [37], which builds upon SNIPER [52] that improved R-CNN training, adds a branch to Faster R-CNN to predict the regions that are likely to contain small objects. Then it applies Faster R-CNN again to each of those regions

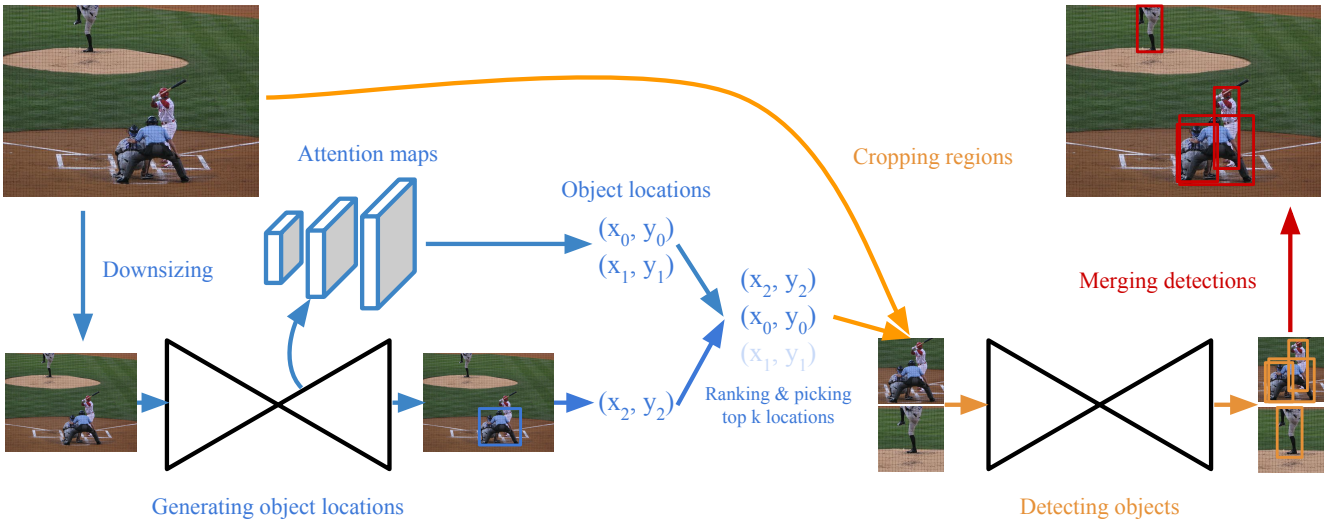

Figure 2: Overview of CornerNet-Saccade. We predict a set of possible object locations from the attention maps and bounding boxes generated on a downsized full image. We zoom into each location and crop a small region around that location. Then we detect objects in each region. We control the efficiency by ranking the object locations and choosing top k locations to process. Finally, we merge the detections by NMS.

by cropping the original pixels. In AutoFocus, there are two kinds of cropping, one that can produce multiple objects (by calling Faster R-CNN as a subroutine), and the other that can produce at most a single object (cropping done within Faster R-CNN). The saccades in AutoFocus are thus *multi-type and mixed*, in the sense that two different types of processing are interleaved.

In contrast, saccades in CornerNet-Saccade are *single-type and multi-object*, in that there is a single type of crop processing and each crop can produce multiple objects at once without additional subcrops. This means that the number of crops processed by CornerNet-Saccade can be much smaller than the number of objects, whereas for R-CNN variants and AutoFocus the number of crops must be no smaller than the number of objects.

Efficient Object Detectors. Other than accuracy [3, 49, 18, 30, 8, 64, 12, 41, 51, 59, 57, 5, 21], many recent works have improved upon the efficiency of detectors since the introduction of R-CNN [11], which applies a ConvNet [24] to 2000 RoIs. Repeatedly applying a ConvNet to the RoIs introduces many redundant computations. SPP [13] and Fast R-CNN [10] address this by applying a ConvNet fully convolutionally on the image and extracting features directly from the feature maps for each RoI. Faster R-CNN [46] further improves efficiency by replacing the low-level vision algorithm with a region proposal network. R-FCN [7] replaces the expensive fully connected sub-detection network with a fully convolutional network, and Light-Head R-CNN [28] reduces the cost in R-FCN by applying separable convolution to reduce the number of channels in the feature maps before RoI pooling. On the other hand, one-stage detectors [34, 43, 9, 44, 31, 56, 23, 20, 60, 63, 48, 62] remove the region pooling step of two-stage detectors.

Efficient Network Architectures. The efficiency of ConvNets is important to many mobile and embedded applications. Much attention [27, 42, 61, 35, 25, 16, 47] has been given to the design of efficient network architectures.

SqueezeNet [19] proposes a fire module to reduce the number of parameters in AlexNet [24] by 50x, while achieving similar performance. MobileNets [15] are a class of lightweight networks that use depth-wise separable convolutions [6], and proposes strategies to achieve a good trade-off between accuracy and latency. PeleeNet [55], in contrast, demonstrates the effectiveness of standard convolutions by introducing an efficient variant of DenseNet [17] consisting of two-way dense layers and a stem block.

Other networks were designed specifically for real-time detection. YOLOv2 [44] incorporates ideas from NIN [29] to design a new variant of VGG [50]. YOLOv3 [45] further improves DarkNet-19 by making the network deeper and introducing skip connections. RFBNet [33] proposes a new module which mimics the receptive field of human vision systems to efficiently gather information across different scales.

3. CornerNet-Saccade

CornerNet-Saccade detects objects within small regions around possible object locations in an image. It uses the downsized full image to predict attention maps and coarse bounding boxes; both suggest possible object locations. CornerNet-Saccade then detects objects by examining the regions centered at the locations in high resolution. It can

also trade accuracy with efficiency by controlling the maximum number of object locations to process per image. An overview of the pipeline is shown in Fig. 2. In this section, we will describe each step in detail.

3.1. Estimating Object Locations

The first step in CornerNet-Saccade is to obtain possible object locations in an image. We use downsized full images to predict attention maps, which indicate both the locations and the coarse scales of the objects at the locations. Given an image, we downsize it to two scales by resizing the longer side of the image to 255 and 192 pixels. The image of size 192 is padded with zeros to the size of 255 so that they can be processed in parallel. There are two reasons for using image at such low resolutions. First, this step should not be a bottleneck in the inference time. Second, the network should easily leverage the context information in the image to predict the attention maps.

For a downsized image, CornerNet-Saccade predicts 3 attention maps, one for small objects, one for medium objects and one for large objects. An object is considered small if the longer side of its bounding box is less than 32 pixels, medium if it is between 32 and 96 pixels, and large if it is greater than 96 pixels[1]. Predicting locations separately for different object sizes gives us finer control over how much CornerNet-Saccade should zoom in at each location. We can zoom in more at small object locations and less at medium object locations.

We predict the attention maps by using feature maps at different scales. The feature maps are obtained from the backbone network in CornerNet-Saccade, which is an hourglass network [39]. Each hourglass module in the network applies several convolution and downsampling layers to downsize the input feature maps. It then upsamples the feature maps back to the original input resolution by multiple convolution and upsampling layers. The feature maps from the upsampling layers are used to predict the attention maps. The feature maps at finer scales are used for smaller objects and the ones at coarser scales are for larger objects. We predict the attention maps by applying a 3×3 Conv-ReLU module followed by a 1×1 Conv-Sigmoid module to each feature map. During testing, we only process locations where scores are above a threshold t, and we set $t = 0.3$ in our experiments.

When CornerNet-Saccade processes the downsized image, it is possible that it detects some of the objects in the image and generates bounding boxes for them. The bounding boxes obtained from the downsized image may not be accurate. Therefore, we also examine the regions in high resolutions to get better bounding boxes.

During training, we set the center location of each bounding box on the corresponding attention map to be positive and the rest to negatives. Then we apply the focal loss with $\alpha = 2$. The biases in the convolution layers that predict the attention maps are set according to [31].

[1]The sizes are w.r.t the input to the network.

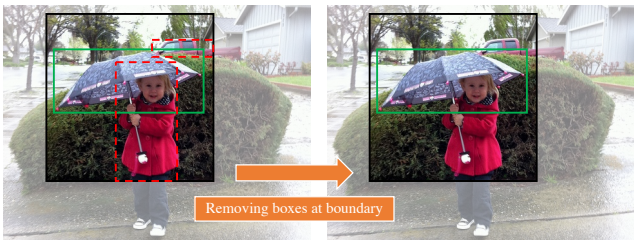

Figure 3: Some objects may not be fully covered by a region. The detector may still generate bounding boxes (red dashed line) for those objects. We remove the bounding boxes which touch the boundaries to avoid such bounding boxes.

3.2. Detecting Objects

CornerNet-Saccade uses the locations obtained from the downsized image to determine where to process. If we directly crop the regions from the downsized image, some objects may become too small to detect accurately. Hence, we should examine the regions at higher resolution based on the scale information obtained in the first step.

For the locations obtained from the attention maps, we can determine different zoom-in scales for different object sizes: s_s for small objects, s_m for medium objects and s_l for large objects. In general, $s_s > s_m > s_l$ because we should zoom in more for smaller objects, so we set $s_s = 4$, $s_m = 2$ and $s_l = 1$. At each possible location (x, y), we enlarge the downsized image by s_i, where $i \in \{s, m, l\}$ depending on the coarse object scale. Then we apply CornerNet-Saccade to a 255×255 window centered at the location.

The locations obtained from the bounding box predictions give more information about the object sizes. We can use the sizes of the bounding boxes to determine zoom-in scales. The scale is determined such that the longer side of the bounding box after zoom-in is 24 for a small object, 64 for a medium object and 192 for a large object.

There are some important implementation details to make processing efficient. First, we process the regions in batch to better utilize the GPU. Second, we keep the original image in GPU memory, and perform resizing and cropping on the GPU to reduce the overhead of transferring image data between CPU and GPU.

After detecting objects at possible object locations, we merge the bounding boxes and remove redundant ones by applying Soft-NMS [2]. When we crop the regions, the regions may include parts of the objects at the crop boundaries as shown in Fig. 3. The detector may generate bounding boxes for those objects, which may not be removed by Soft-NMS as they may have low overlaps with the bounding

Figure 4: When the objects are close to each other, we may generate regions that highly overlap with each other. Processing either one of them is likely to detect objects in all highly overlapping regions. We suppress redundant regions to improve efficiency.

boxes of the full objects. Hence, we remove the bounding boxes which touch the crop boundary. During training, we apply the same training losses in CornerNet to train the network to predict corner heatmaps, embeddings and offsets.

3.3. Trading Accuracy with Efficiency

We can trade accuracy with efficiency by controlling the maximum number of object locations to process per image. To achieve a good accuracy and efficiency trade-off, we prioritize the locations that are more likely to contain objects. Therefore, after we obtain the object locations, we rank them by their scores and prioritize locations obtained from the bounding boxes. Given the maximum number of crops to process k_{max}, we detect objects in the top k_{max} object locations.

3.4. Suppressing Redundant Object Locations

When objects are close to each other, we may generate regions that highly overlap with each other as shown in Fig. 4. It is undesirable to process both regions as processing either one of them is likely to detect objects in the other.

We adopt a procedure similar to NMS to remove redundant locations. First, we rank the object locations, prioritizing locations from bounding boxes over locations from the attention maps. We then keep the best object location and remove the locations that are close to the best location. We repeat the procedure until no object locations are left.

3.5. Backbone Network

We design a new hourglass backbone network that works better in CornerNet-Saccade. The new hourglass network consists of 3 hourglass modules and has a depth of 54 layers, while Hourglass-104 in CornerNet consists of 2 hourglass modules and has a depth of 104. We refer to the new backbone as Hourglass-54.

Each hourglass module in Hourglass-54 has fewer parameters and is shallower than the one in Hourglass-104. Following the downsizing strategy in Hourglass-104, we downsize the feature by stride 2. We apply one residual module [14] after each downsampling layer and in each skip connection. Each hourglass module downsizes the input features 3 times and increases the number of channels along the way (384, 384, 512). There is one residual module with 512 channels in the middle of the module, and one residual module after each upsampling layer. We also downsize the image twice before the hourglass modules.

Following the common practice of training an hourglass network, we also add intermediate supervisions during training. During testing, we only use the predictions from the last hourglass module in the network.

3.6. Training Details

We use Adam [22] to optimize both the losses for the attention maps and object detection, and use the same training hyperparameters found in CornerNet. The input size to the network is 255×255, which is also the input resolution during inference. We train the network with a batch size of 48 on four 1080Ti GPUs. In order to avoid over-fitting, we adopt the data augmentation techniques used in CornerNet. When we randomly crop a region around an object, the object is either placed randomly or at the center with some random offset. This ensures that training and testing are consistent as the network detects objects within the crops centered at object locations.

4. CornerNet-Squeeze

4.1. Overview

In contrast to CornerNet-Saccade, which focuses on a subset of the pixels to reduce the amount of processing, CornerNet-Squeeze explores an alternative approach of reducing the amount of processing per pixel. In CornerNet, most of the computational resources are spent on Hourglass-104. Hourglass-104 is built from residual blocks which consists of two 3×3 convolution layers and a skip connection. Although Hourglass-104 achieves competitive performance, it is expensive in terms of number of parameters and inference time. To reduce the complexity of Hourglass-104, we incorporate ideas from SqueezeNet [19] and MobileNets [15] to design a lightweight hourglass architecture.

4.2. Ideas from SqueezeNet and MobileNets

SqueezeNet proposes three strategies to reduce network complexity: (1) replacing 3×3 kernels with 1×1 kernels; (2) decreasing input channels to 3×3 kernels; (3) downsampling late. The building block of SqueezeNet, the fire module, encapsulates the first two ideas. The *fire module* first reduces the number of input channels with a *squeeze* layer consisting of 1×1 filters. Then, it feeds the result through an *expand* layer consisting of a mixture of 1×1 and 3×3 filters.

Figure 5: Qualitative examples on COCO validation set.

Based on the insights provided by SqueezeNet, we use the fire module in CornerNet-Squeeze instead of the residual block. Furthermore, inspired by the success of MobileNets, we replace the 3×3 standard convolution in the second layer with a 3×3 depth-wise separable convolution, which further improves inference time. Tab. 1 shows a detail comparison between the residual block in CornerNet and the new fire module in CornerNet-Squeeze.

We do not explore the third idea in SqueezeNet. Since the hourglass network has a symmetrical structure, delayed downsampling results in higher resolution feature maps during the upsampling. Performing convolution on high resolution feature maps is computationally expensive, which would prevent us from achieving real-time detection.

Input	Operator	Output
Residual block in CornerNet		
$h \times w \times k$	3×3 Conv, ReLU	$h \times w \times k'$
$h \times w \times k'$	3×3 Conv, ReLU	$h \times w \times k'$
Fire module in CornerNet-Squeeze		
$h \times w \times k$	1×1 Conv	$h \times w \times \frac{k'}{2}$
$h \times w \times \frac{k'}{2}$	1×1 Conv + 3×3 Dwise, ReLU	$h \times w \times k'$

Table 1: Comparison between the residual block in CornerNet and the fire module in CornerNet-Squeeze.

Other than replacing the residual blocks, we also make a few more modifications. We reduce the maximum feature map resolution of the hourglass modules by adding one more downsampling layer before the hourglass modules, and remove one downsampling layer in each hourglass module. CornerNet-Squeeze correspondingly downsizes the image three times before the hourglass module, whereas CornerNet downsizes the image twice. We replace the 3×3 filters with 1×1 filters in the prediction modules of CornerNet. Finally, we replace the nearest neighbor upsampling in the hourglass network with transpose convolution with a 4×4 kernel.

4.3. Training Details

We use the same training losses and hyperparameters of CornerNet to train CornerNet-Squeeze. The only change is the batch size. Downsizing the image one more time prior to the hourglass modules reduces the memory usage by 4 times under the same image resolution in CornerNet-Squeeze. We are able to train the network with a batch size of 55 on four 1080Ti GPUs (13 images on the master GPU and 14 images per GPU for the rest of the GPUs).

5. Experiments

5.1. Implementation Details

CornerNet-Lite is implemented in PyTorch [40]. We use COCO [32] to evaluate CornerNet-Lite and compare it with other detectors. In COCO, there are 80 object categories, 115k images for training, 5k images for validation and 20k images for testing.

To measure the inference time, for each detector, we start the timer as soon as it finishes reading the image and stop the timer as soon as it obtains the final bounding boxes. The hardware configuration may affect the inference time. To provide fair comparisons between different detectors, we measure the inference times on the same machine with a 1080Ti GPU and an Intel Core i7-7700k CPU.

5.2. Accuracy and Efficiency Trade-offs

We compare the accuracy-efficiency trade-offs of CornerNet-Lite with three state-of-the-art object detectors, including YOLOv3 [45], RetinaNet[2] [31] and CornerNet [26], on the *validation set*. The accuracy and efficiency trade-off curves are shown in Fig. 6.

We evaluate CornerNet-Saccade under different k_{max}, ranging from 1 to 30. For RetinaNet, we evaluate at different single scale settings, including 300, 400, 500, 600,

[2]We use the X-101-64x4d-FPN model available on https://github.com/facebookresearch/Detectron/blob/master/MODEL_ZOO.md

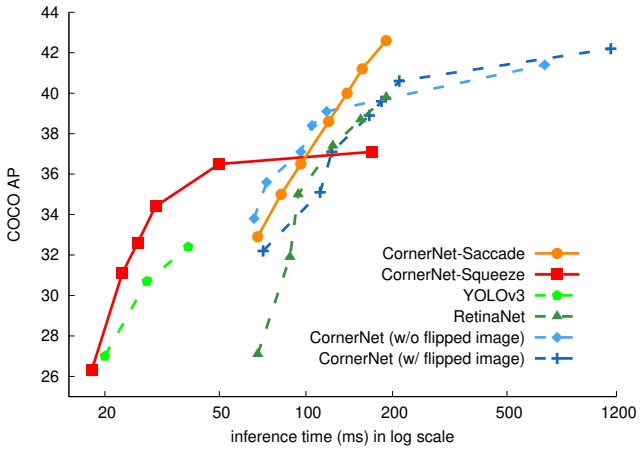

Figure 6: Both CornerNet-Saccade and CornerNet-Squeeze achieve better trade-offs than other state-of-the-art one-stage detectors. The inference time is in log scale.

700, and 800 (the default scale). For CornerNet, we evaluate at different single scales of the original image resolutions, including 0.5, 0.6, 0.7, 0.8, 0.9 and 1 (original image resolution). We also test it under the default multi-scale setting, and without flipped image. CornerNet-Saccade achieves a better accuracy and efficiency trade-off (42.6% at 190ms) than both RetinaNet (39.8% at 190ms) and CornerNet (40.6% at 213ms). Fig. 5 shows some qualitative examples comparing CornerNet-Saccade and CornerNet.

Following the default settings of YOLOv3, we evaluate YOLOv3 at 3 single image scales (320, 416 and 608). Similarly, we also evaluate CornerNet-Squeeze at different single scales (0.5, 0.6, 0.7, 0.8, 0.9, 1). CornerNet-Squeeze achieves a better accuracy and efficiency (34.4% at 30ms) trade-off than YOLOv3 (32.4% at 39ms). We further improve the accuracy of CornerNet-Squeeze by running it on both flipped and original images, which improves its AP to 36.5% at 50ms and still achieves a good trade-off. When we test CornerNet-Squeeze under multi-scale setting, we observe only a 0.6% improvement in AP but the inference time increases to 170ms.

5.3. CornerNet-Saccade Results

Training Efficiency CornerNet-Saccade not only improves the efficiency in testing but also in training. We are able to train CornerNet-Saccade on only four 1080Ti GPUs with a total of 44GB GPU memory, while CornerNet requires ten Titan X (PASCAL) GPUs with a total of 120GB GPU memory. We reduce the memory usage by more than 60%. Neither CornerNet nor CornerNet-Saccade uses mixed precision training [36].

Error Analysis The attention maps are important to CornerNet-Saccade. If the attention maps are inaccurate, CornerNet-Saccade will miss objects in the image. To give a better understanding of the attention map quality, we replace the predicted attention maps with the ground-truth ones. This improves the AP of CornerNet-Saccade from 42.6% to 50.3% on the validation set, showing there is ample room for improving the attention maps.

Detector	GPU	Quantity	Total Mem
CornerNet	Titan X (PASCAL)	10	120GB
CornerNet-Saccade	1080Ti	4	44GB

Table 2: CornerNet-Saccade saves more than 60% GPU memory and requires only 4 GPUs to train, while it achieves results competitive to CornerNet.

	AP	AP^s	AP^m	AP^l
CornerNet-Saccade	42.6	25.5	44.3	58.4
+ gt attention	50.3	32.3	53.4	65.3

Table 3: The quality of the attention maps is a bottleneck in CornerNet-Saccade.

Performance Analysis of Hourglass-54 We introduce a new hourglass, Hourglass-54, in CornerNet-Saccade, and perform two experiments to better understand the performance contribution of Hourglass-54. First, we train CornerNet-Saccade with Hourglass-104 instead of Hourglass-54. Second, we train CornerNet with Hourglass-54 instead of Hourglass-104. For the second experiment, due to limited resources, we train both networks with a batch size of 15 on four 1080Ti GPUs and we follow the training details in CornerNet [26].

Tab. 4 shows that CornerNet-Saccade with Hourglass-54 (42.6% AP) is more accurate than with Hourglass-104 (41.4%). To investigate the difference in performance, we evaluate the quality of both the attention maps and bounding boxes. First, predicting the attention maps can be seen as a binary classification problem, where the object locations are positives and the rest are negatives. We measure the quality of the attention maps by average precision, denoted as AP^{att}. Hourglass-54 achieves an AP^{att} of 42.7%, while Hourglass-104 achieves 40.1%, suggesting that Hourglass-54 is better at predicting attention maps.

Second, to study the quality of bounding boxes from each network, we replace the predicted attention maps with the ground-truth attention maps, and also train CornerNet with Hourglass-54. With the ground-truth attention maps, CornerNet-Saccade with Hourglass-54 achieves an AP of 50.3% while CornerNet-Saccade with Hourglass-104 achieves an AP of 48.9%. CornerNet with Hourglass-54 achieves an AP of 37.2%, while Hourglass-104 achieves 38.2%. The results suggest that Hourglass-54 produces better bounding boxes when combined with saccade.

	AP	AP^s	AP^m	AP^l	AP^{att}
CornerNet-Saccade w/ HG-54	42.6	25.5	44.3	58.4	42.7
+ gt attention	50.3	32.3	53.4	65.3	-
CornerNet-Saccade w/ HG-104	41.4	23.8	43.5	57.1	40.1
+ gt attention	48.9	32.4	51.8	62.6	-
CornerNet w/ HG-54	37.2	18.4	40.3	49.0	-
CornerNet w/ HG-104	38.2	18.6	40.9	50.1	-

Table 4: CornerNet-Saccade with Hourglass-54 produces better results when combined with saccade.

5.4. CornerNet-Squeeze Results

Comparison with YOLOv3 We compare CornerNet-Squeeze with one of the widely used real-time detectors, YOLOv3 [45], in Tab. 5. YOLOv3 is implemented in C and also provides a Python API, which adds a 10ms overhead to the inference time. On the other hand, CornerNet-Squeeze is implemented in Python and still faster than the C version of YOLOv3. There is a potential speed-up if we implement CornerNet-Squeeze purely in C.

	Language	Time	AP
YOLOv3	C	39ms	33.0
YOLOv3	Python	49ms	33.0
CornerNet-Squeeze	Python	**30ms**	**34.4**

Table 5: COCO Test AP. CornerNet-Squeeze is faster and more accurate than YOLOv3.

Ablation Study We study each major change in CornerNet-Squeeze to understand its contribution to the inference time and AP. To conserve GPU resources, each model is only trained for 250k iterations, following the details in Sec. 4.3. With the extra downsize before the hourglass modules, we are able to train the network with a batch size of 55 (Sec. 4.3), while we can only train CornerNet with a batch size of 15 on four 1080Ti GPUs. We just provide CornerNet APs at 250k in Tab. 6 as a reference.

In CornerNet, there are two downsampling layers before the hourglass modules. If we add one more downsampling layer, we can reduce the inference time from 114ms to 46ms. Replacing the residual blocks with fire modules saves 11ms and using 1×1 kernel in predicting layers saves another 2ms without any loss in performance. Finally, we use transpose convolution as it improves the AP by 0.5% with a small increase in inference time.

	Time	AP
CornerNet	211ms	31.4*
+ w/o flipped image	111ms	29.7*
+ one extra downsampling before HG modules	41ms	33.0
+ replace residual blocks with fire modules	31ms	29.8
+ replace 3×3 with 1×1 conv in prediction layers	28ms	29.8
+ upsample using transpose conv (CornerNet-Squeeze)	30ms	30.3

Table 6: Ablation study on CornerNet-Squeeze. *Note that CornerNet is trained with a much smaller batch size.

5.5. CornerNet-Squeeze-Saccade Results

We try combining CornerNet-Squeeze with saccades to further improve the efficiency. However, we find that CornerNet-Squeeze-Saccade does not outperform CornerNet-Squeeze. On the validation set, CornerNet-Squeeze achieves an AP of 34.4%, while CornerNet-Squeeze-Saccade with $k_{max} = 30$ achieves 32.7%. If we replace the predicted attention map with the ground-truth attention map (i.e. the object locations are known), we improve the AP of CornerNet-Squeeze-Saccade to 38.0%, outperforming CornerNet-Squeeze.

The results suggest that saccade can only help if the attention maps are sufficiently accurate. Due to its ultra-compact architecture, CornerNet-Squeeze-Saccade does not have enough capacity to detect objects and predict attention maps simultaneously. Furthermore, CornerNet-Squeeze only operates on single scale images, which provides much less room for CornerNet-Squeeze-Saccade to save. CornerNet-Squeeze-Saccade may process more number of pixels than CornerNet-Squeeze, slowing down the inference time.

	Time	AP	AP^s	AP^m	AP^l
CornerNet-Squeeze-Saccade	61ms	32.7	17.3	32.6	47.1
+ gt attention	-	38.0	24.4	39.3	50.2
CornerNet-Squeeze	30ms	34.4	14.8	36.9	49.5

Table 7: CornerNet-Squeeze-Saccade runs slower and is less accurate than Cornernet-Squeeze. Saccade only helps if the attention maps are sufficiently accurate. But CornerNet-Squeeze-Saccade does not have enough capacity to predict attention maps and detect objects due to its ultra-compact structure.

5.6. COCO Test AP

We also compare CornerNet-Lite with CornerNet and YOLOv3 on COCO test set in Tab. 8. CornerNet-Squeeze is faster and more accurate than YOLOv3. CornerNet-Saccade is more accurate than CornerNet at multi-scales and 6 times faster.

	Time	AP	AP^s	AP^m	AP^l
YOLOv3	39ms	33.0	18.3	35.4	41.9
CornerNet-Squeeze	30ms	34.4	13.7	36.5	47.4
CornerNet (single)	211ms	40.6	19.1	42.8	54.3
CornerNet (multi)	1147ms	42.2	20.7	44.8	56.6
CornerNet-Saccade	190ms	43.2	24.4	44.6	57.3

Table 8: CornerNet-Lite versus CornerNet and YOLOv3 on COCO test set.

6. Conclusions

We propose CornerNet-Lite which is a combination of two efficient variant of CornerNet: CornerNet-Saccade and

CornerNet-Squeeze. Together these contributions for the first time reveal the potential of keypoint-based detection to be useful for applications requiring processing efficiency.

Acknowledgements This work is partially supported by a DARPA grant FA8750-18-2-0019 and an IARPA grant D17PC00343.

References

[1] A. T. Bahill, M. R. Clark, and L. Stark. The main sequence, a tool for studying human eye movements. *Mathematical Biosciences*, 24(3-4):191–204, 1975.

[2] N. Bodla, B. Singh, R. Chellappa, and L. S. Davis. Soft-nms–improving object detection with one line of code. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5561–5569, 2017.

[3] Z. Cai, Q. Fan, R. S. Feris, and N. Vasconcelos. A unified multi-scale deep convolutional neural network for fast object detection. In *European conference on computer vision*, pages 354–370. Springer, 2016.

[4] Z. Cai and N. Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6154–6162, 2018.

[5] B. Cheng, Y. Wei, H. Shi, R. Feris, J. Xiong, and T. Huang. Revisiting rcnn: On awakening the classification power of faster rcnn. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 453–468, 2018.

[6] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[7] J. Dai, Y. Li, K. He, and J. Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016.

[8] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei. Deformable convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 764–773, 2017.

[9] C.-Y. Fu, W. Liu, A. Ranga, A. Tyagi, and A. C. Berg. Dssd: Deconvolutional single shot detector. *arXiv preprint arXiv:1701.06659*, 2017.

[10] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[11] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[12] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[13] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015.

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[15] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[16] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[17] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[18] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7310–7311, 2017.

[19] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[20] J. Jeong, H. Park, and N. Kwak. Enhancement of ssd by concatenating feature maps for object detection. *arXiv preprint arXiv:1705.09587*, 2017.

[21] B. Jiang, R. Luo, J. Mao, T. Xiao, and Y. Jiang. Acquisition of localization confidence for accurate object detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–799, 2018.

[22] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[23] T. Kong, F. Sun, A. Yao, H. Liu, M. Lu, and Y. Chen. Ron: Reverse connection with objectness prior networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5936–5944, 2017.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[25] K. A. Laube and A. Zell. Shufflenasnets: Efficient cnn models through modified efficient neural architecture search. *arXiv preprint arXiv:1812.02975*, 2018.

[26] H. Law and J. Deng. Cornernet: Detecting objects as paired keypoints. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 734–750, 2018.

[27] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[28] Z. Li, C. Peng, G. Yu, X. Zhang, Y. Deng, and J. Sun. Light-head r-cnn: In defense of two-stage object detector. *arXiv preprint arXiv:1711.07264*, 2017.

[29] M. Lin, Q. Chen, and S. Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[30] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision*

and Pattern Recognition, pages 2117–2125, 2017.

[31] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[32] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[33] S. Liu, D. Huang, et al. Receptive field block net for accurate and fast object detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 385–400, 2018.

[34] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[35] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 116–131, 2018.

[36] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

[37] M. Najibi, B. Singh, and L. S. Davis. Autofocus: Efficient multi-scale inference. *arXiv preprint arXiv:1812.01600*, 2018.

[38] A. Newell, Z. Huang, and J. Deng. Associative embedding: End-to-end learning for joint detection and grouping. In *Advances in Neural Information Processing Systems*, pages 2277–2287, 2017.

[39] A. Newell, K. Yang, and J. Deng. Stacked hourglass networks for human pose estimation. In *European Conference on Computer Vision*, pages 483–499. Springer, 2016.

[40] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[41] C. Peng, T. Xiao, Z. Li, Y. Jiang, X. Zhang, K. Jia, G. Yu, and J. Sun. Megdet: A large mini-batch object detector. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6181–6189, 2018.

[42] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[43] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[44] J. Redmon and A. Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[45] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[46] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[47] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[48] Z. Shen, Z. Liu, J. Li, Y.-G. Jiang, Y. Chen, and X. Xue. Dsod: Learning deeply supervised object detectors from scratch. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1919–1927, 2017.

[49] A. Shrivastava, R. Sukthankar, J. Malik, and A. Gupta. Beyond skip connections: Top-down modulation for object detection. *arXiv preprint arXiv:1612.06851*, 2016.

[50] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[51] B. Singh and L. S. Davis. An analysis of scale invariance in object detection snip. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3578–3587, 2018.

[52] B. Singh, M. Najibi, and L. S. Davis. Sniper: Efficient multi-scale training. In *Advances in Neural Information Processing Systems*, pages 9333–9343, 2018.

[53] L. Tychsen-Smith and L. Petersson. Denet: Scalable real-time object detection with directed sparse sampling. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 428–436, 2017.

[54] P. Viola, M. Jones, et al. Rapid object detection using a boosted cascade of simple features. *CVPR (1)*, 1:511–518, 2001.

[55] R. J. Wang, X. Li, and C. X. Ling. Pelee: A real-time object detection system on mobile devices. In *Advances in Neural Information Processing Systems*, pages 1963–1972, 2018.

[56] X. Wang, K. Chen, Z. Huang, C. Yao, and W. Liu. Point linking network for object detection. *arXiv preprint arXiv:1706.03646*, 2017.

[57] H. Xu, X. Lv, X. Wang, Z. Ren, N. Bodla, and R. Chellappa. Deep regionlets for object detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 798–814, 2018.

[58] A. L. Yarbus. *Eye movements and vision*. Springer, 2013.

[59] Y. Zhai, J. Fu, Y. Lu, and H. Li. Feature selective networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4139–4147, 2018.

[60] S. Zhang, L. Wen, X. Bian, Z. Lei, and S. Z. Li. Single-shot refinement neural network for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4203–4212, 2018.

[61] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018.

[62] Z. Zhang, S. Qiao, C. Xie, W. Shen, B. Wang, and A. L. Yuille. Single-shot object detection with enriched semantics. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5813–5821, 2018.

[63] Q. Zhao, T. Sheng, Y. Wang, Z. Tang, Y. Chen, L. Cai, and H. Ling. M2det: A single-shot object detector based on multi-level feature pyramid network. *arXiv preprint arXiv:1811.04533*, 2018.

[64] X. Zhu, H. Hu, S. Lin, and J. Dai. Deformable convnets v2: More deformable, better results. *arXiv preprint arXiv:1811.11168*, 2018.