# Deformable Convolutional Networks

Jifeng Dai* Haozhi Qi*,† Yuwen Xiong*,† Yi Li*,† Guodong Zhang*,† Han Hu Yichen Wei

Microsoft Research Asia

{jifdai,v-haoq,v-yuxio,v-yii,v-guodzh,hanhu,yichenw}@microsoft.com

## Abstract

*Convolutional neural networks (CNNs) are inherently limited to model geometric transformations due to the fixed geometric structures in their building modules. In this work, we introduce two new modules to enhance the transformation modeling capability of CNNs, namely, deformable convolution and deformable RoI pooling. Both are based on the idea of augmenting the spatial sampling locations in the modules with additional offsets and learning the offsets from the target tasks, without additional supervision. The new modules can readily replace their plain counterparts in existing CNNs and can be easily trained end-to-end by standard back-propagation, giving rise to* deformable convolutional networks. *Extensive experiments validate the performance of our approach. For the first time, we show that learning dense spatial transformation in deep CNNs is effective for sophisticated vision tasks such as object detection and semantic segmentation. The code is released at https://github.com/msracver/Deformable-ConvNets.*

## 1. Introduction

A key challenge in visual recognition is how to accommodate geometric variations or model geometric transformations in object scale, pose, viewpoint, and part deformation. In general, there are two ways. The first is to build the training datasets with sufficient desired variations. This is usually realized by augmenting the existing data samples, *e.g.*, by affine transformation. Robust representations can be learned from the data, but usually at the cost of expensive training and complex model parameters. The second is to use transformation-invariant features and algorithms. This category subsumes many well known techniques, such as SIFT (scale invariant feature transform) [42] and sliding window based object detection paradigm.

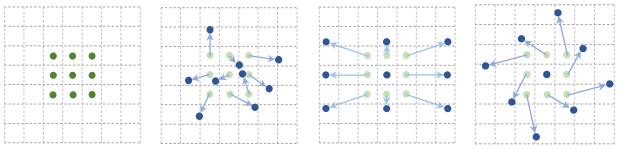
There are two drawbacks in above ways. First, the geometric transformations are assumed fixed and known. Such prior knowledge is used to augment the data, and design the features and algorithms. This assumption prevents generalization to new tasks possessing unknown geometric transformations, which are not properly modeled. Second, handcrafted design of invariant features and algorithms could be difficult or infeasible for overly complex transformations, even when they are known.

Recently, convolutional neural networks (CNNs) [35] have achieved significant success for visual recognition tasks, such as image classification [31], semantic segmentation [41], and object detection [16]. Nevertheless, they still share the above two drawbacks. Their capability of modeling geometric transformations mostly comes from the extensive data augmentation, the large model capacity, and some simple hand-crafted modules (*e.g.*, max-pooling [1] for small translation-invariance).

In short, CNNs are inherently limited to model large, unknown transformations. The limitation originates from the fixed geometric structures of CNN modules: a convolution unit samples the input feature map at fixed locations; a pooling layer reduces the spatial resolution at a fixed ratio; a RoI (region-of-interest) pooling layer separates a RoI into fixed spatial bins, etc. There lacks internal mechanisms to handle the geometric transformations. This causes noticeable problems. *For one example*, the receptive field sizes of all activation units in the same CNN layer are the same. This is undesirable for high level CNN layers that encode the semantics over spatial locations. Because different locations may correspond to objects with different scales or deformation, adaptive determination of scales or receptive field sizes is desirable for visual recognition with fine localization, *e.g.*, semantic segmentation using fully convolutional networks [41]. *For another example*, while object detection has seen significant and rapid progress [16, 52, 15, 47, 46, 40, 7] recently, all approaches still rely on the primitive bounding box based feature extraction. This is clearly sub-optimal, especially for non-rigid objects.

In this work, we introduce two new modules that greatly enhance CNNs' capability of modeling geometric transfor-

*Equal contribution. †This work is done when Haozhi Qi, Yuwen Xiong, Yi Li and Guodong Zhang are interns at Microsoft Research Asia



(a) (b) (c) (d)

Figure 1: Illustration of the sampling locations in $3 \times 3$ standard and deformable convolutions. (a) regular sampling grid (green points) of standard convolution. (b) deformed sampling locations (dark blue points) with augmented offsets (light blue arrows) in deformable convolution. (c)(d) are special cases of (b), showing that the deformable convolution generalizes various transformations for scale, (anisotropic) aspect ratio and rotation.

mations. The first is *deformable convolution*. It adds 2D offsets to the regular grid sampling locations in the standard convolution. It enables free form deformation of the sampling grid. It is illustrated in Figure 1. The offsets are learned from the preceding feature maps, via additional convolutional layers. Thus, the deformation is conditioned on the input features in a local, dense, and adaptive manner.

The second is *deformable RoI pooling*. It adds an offset to each bin position in the regular bin partition of the previous RoI pooling [15, 7]. Similarly, the offsets are learned from the preceding feature maps and the RoIs, enabling adaptive part localization for objects with different shapes.

Both modules are light weight. They add small amount of parameters and computation for the offset learning. They can readily replace their plain counterparts in deep CNNs and can be easily trained end-to-end with standard back-propagation. The resulting CNNs are called *deformable convolutional networks*, or *deformable ConvNets*.

Our approach shares similar high level spirit with spatial transform networks [26] and deformable part models [11]. They all have internal transformation parameters and learn such parameters purely from data. A key difference in deformable ConvNets is that they deal with dense spatial transformations in a simple, efficient, deep and end-to-end manner. In Section 3.1, we discuss in details the relation of our work to previous works and analyze the superiority of deformable ConvNets.

## 2. Deformable Convolutional Networks

The feature maps and convolution in CNNs are 3D. Both deformable convolution and RoI pooling modules operate on the 2D spatial domain. The operation remains the same across the channel dimension. Without loss of generality, the modules are described in 2D here for notation clarity. Extension to 3D is straightforward.

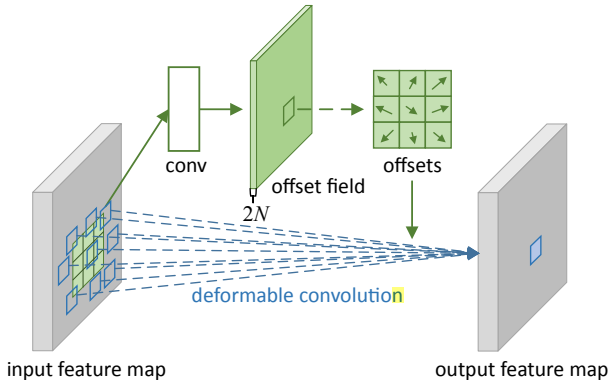



Figure 2: Illustration of $3 \times 3$ deformable convolution.

### 2.1. Deformable Convolution

The 2D convolution consists of two steps: 1) sampling using a regular grid $\mathcal{R}$ over the input feature map $\mathbf{x}$; 2) summation of sampled values weighted by $\mathbf{w}$. The grid $\mathcal{R}$ defines the receptive field size and dilation. For example,

$$\mathcal{R} = \{(-1,-1),(-1,0),\ldots,(0,1),(1,1)\}$$

defines a $3 \times 3$ kernel with dilation 1.

For each location $\mathbf{p}_0$ on the output feature map $\mathbf{y}$, we have

$$\mathbf{y}(\mathbf{p}_0) = \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n), \quad (1)$$

where $\mathbf{p}_n$ enumerates the locations in $\mathcal{R}$.

In deformable convolution, the regular grid $\mathcal{R}$ is augmented with offsets $\{\Delta\mathbf{p}_n | n = 1, ..., N\}$, where $N = |\mathcal{R}|$. Eq. (1) becomes

$$\mathbf{y}(\mathbf{p}_0) = \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n + \Delta\mathbf{p}_n). \quad (2)$$

Now, the sampling is on the irregular and offset locations $\mathbf{p}_n + \Delta\mathbf{p}_n$. As the offset $\Delta\mathbf{p}_n$ is typically fractional, Eq. (2) is implemented via bilinear interpolation as

$$\mathbf{x}(\mathbf{p}) = \sum_{\mathbf{q}} G(\mathbf{q}, \mathbf{p}) \cdot \mathbf{x}(\mathbf{q}), \quad (3)$$

where $\mathbf{p}$ denotes an arbitrary (fractional) location ($\mathbf{p} = \mathbf{p}_0 + \mathbf{p}_n + \Delta\mathbf{p}_n$ for Eq. (2)), $\mathbf{q}$ enumerates all integral spatial locations in the feature map $\mathbf{x}$, and $G(\cdot,\cdot)$ is the bilinear interpolation kernel. Note that $G$ is two dimensional. It is separated into two one dimensional kernels as

$$G(\mathbf{q}, \mathbf{p}) = g(q_x, p_x) \cdot g(q_y, p_y), \quad (4)$$

where $g(a, b) = max(0, 1 - |a - b|)$. Eq. (3) is fast to compute as $G(\mathbf{q}, \mathbf{p})$ is non-zero only for a few $\mathbf{q}$s.

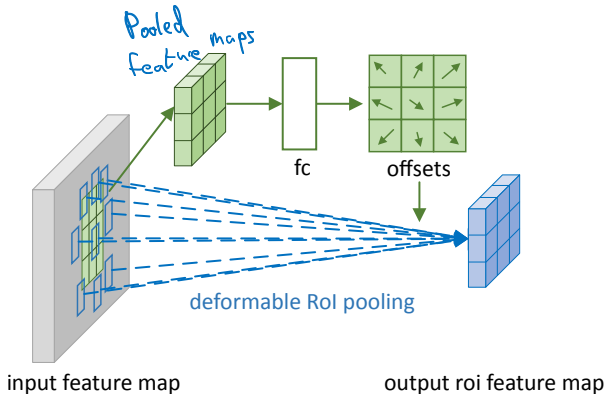



Figure 3: Illustration of $3 \times 3$ deformable RoI pooling.

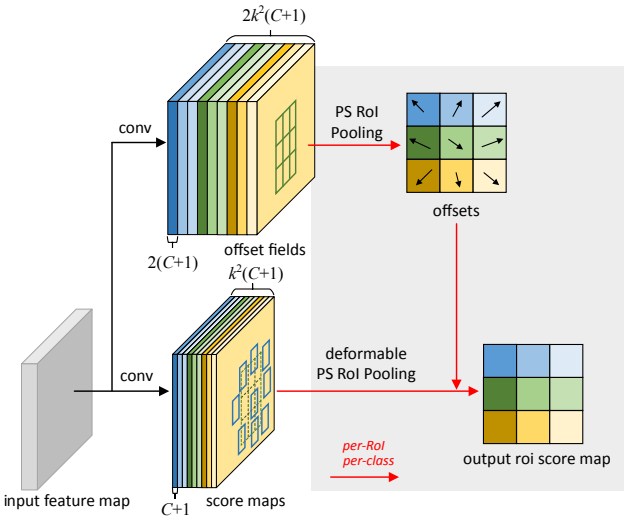



Figure 4: Illustration of $3 \times 3$ deformable PS RoI pooling.

As illustrated in Figure 2, the offsets are obtained by applying a convolutional layer over the same input feature map. The convolution kernel is of the same spatial resolution and dilation as those of the current convolutional layer (*e.g.*, also $3 \times 3$ with dilation 1 in Figure 2). The output offset fields have the same spatial resolution with the input feature map. The channel dimension $2N$ corresponds to $N$ 2D offsets. During training, both the convolutional kernels for generating the output features and the offsets are learned simultaneously. To learn the offsets, the gradients are back-propagated through the bilinear operations in Eq. (3) and Eq. (4). It is detailed in appendix A.

## 2.2. Deformable RoI Pooling

RoI pooling is used in all region proposal based object detection methods [16, 15, 47, 7]. It converts an input rectangular region of arbitrary size into fixed size features.

**RoI Pooling [15]** Given the input feature map $\mathbf{x}$ and a RoI of size $w \times h$ and top-left corner $\mathbf{p}_0$, RoI pooling divides the RoI into $k \times k$ ($k$ is a free parameter) bins and outputs a $k \times k$ feature map $\mathbf{y}$. For $(i, j)$-th bin ($0 \le i, j < k$), we have

$$\mathbf{y}(i,j) = \sum_{\mathbf{p} \in bin(i,j)} \mathbf{x}(\mathbf{p}_0 + \mathbf{p})/n_{ij}, \tag{5}$$

where $n_{ij}$ is the number of pixels in the bin. The $(i, j)$-th bin spans $\lfloor i\frac{w}{k} \rfloor \le p_x < \lceil (i+1)\frac{w}{k} \rceil$ and $\lfloor j\frac{h}{k} \rfloor \le p_y < \lceil (j+1)\frac{h}{k} \rceil$.

Similarly as in Eq. (2), in deformable RoI pooling, offsets $\{\Delta\mathbf{p}_{ij} | 0 \le i, j < k\}$ are added to the spatial binning positions. Eq.(5) becomes

$$\mathbf{y}(i,j) = \sum_{\mathbf{p} \in bin(i,j)} \mathbf{x}(\mathbf{p}_0 + \mathbf{p} + \Delta\mathbf{p}_{ij})/n_{ij}. \tag{6}$$

Typically, $\Delta\mathbf{p}_{ij}$ is fractional. Eq. (6) is implemented by bilinear interpolation via Eq. (3) and (4).

Figure 3 illustrates how to obtain the offsets. Firstly, RoI pooling (Eq. (5)) generates the pooled feature maps. From the maps, a fc layer generates the *normalized* offsets $\Delta\widehat{\mathbf{p}}_{ij}$, which are then transformed to the offsets $\Delta\mathbf{p}_{ij}$ in Eq. (6) by element-wise product with the RoI's width and height, as $\Delta\mathbf{p}_{ij} = \gamma \cdot \Delta\widehat{\mathbf{p}}_{ij} \circ (w, h)$. Here $\gamma$ is a pre-defined scalar to modulate the magnitude of the offsets. It is empirically set to $\gamma = 0.1$. The offset normalization is necessary to make the offset learning invariant to RoI size. The fc layer is learned by back-propagation, as detailed in appendix A.

**Position-Sensitive (PS) RoI Pooling [7]** It is fully convolutional and different from RoI pooling. Through a conv layer, all the input feature maps are firstly converted to $k^2$ *score maps* for each object class (totally $C + 1$ for $C$ object classes), as illustrated in the bottom branch in Figure 4. Without need to distinguish between classes, such score maps are denoted as $\{\mathbf{x}_{i,j}\}$ where $(i, j)$ enumerates all bins. Pooling is performed on these score maps. The output value for $(i, j)$-th bin is obtained by summation from one score map $\mathbf{x}_{i,j}$ corresponding to that bin. In short, the difference from RoI pooling in Eq.(5) is that a general feature map $\mathbf{x}$ is replaced by a specific positive-sensitive score map $\mathbf{x}_{i,j}$.

In deformable PS RoI pooling, the only change in Eq. (6) is that $\mathbf{x}$ is also modified to $\mathbf{x}_{i,j}$. However, the offset learning is different. It follows the "fully convolutional" spirit in [7], as illustrated in Figure 4. In the top branch, a conv layer generates the full spatial resolution offset fields. For each RoI (also for each class), PS RoI pooling is applied on such fields to obtain *normalized* offsets $\Delta\widehat{\mathbf{p}}_{ij}$, which are then transformed to the real offsets $\Delta\mathbf{p}_{ij}$ in the same way as in deformable RoI pooling described above.

## 2.3. Deformable ConvNets

Both deformable convolution and RoI pooling modules have the same input and output as their plain versions.

Hence, they can readily replace their plain counterparts in existing CNNs. In the training, these added conv and fc layers for offset learning are initialized with zero weights. Their learning rates are set to $\beta$ times ($\beta = 1$ by default, and $\beta = 0.01$ for the fc layer in Faster R-CNN) of the learning rate for the existing layers. They are trained via back propagation through the bilinear interpolation operations in Eq. (3) and Eq. (4). The resulting CNNs are called *deformable ConvNets*.

To integrate deformable ConvNets with the state-of-the-art CNN architectures, we note that these architectures consist of two stages. First, a deep fully convolutional network generates feature maps over the whole input image. Second, a shallow task specific network generates results from the feature maps. We elaborate the two steps below.

**Deformable Convolution for Feature Extraction** We adopt two state-of-the-art architectures for feature extraction: ResNet-101 [22] and a modifed version of Inception-ResNet [51]. Both are pre-trained on ImageNet [8] classification dataset.

The original Inception-ResNet is designed for image recognition. It has a feature misalignment issue and problematic for dense prediction tasks. It is modified to fix the alignment problem [20]. The modified version is dubbed as "Aligned-Inception-ResNet" and is detailed in appendix B.

Both models consist of several convolutional blocks, an average pooling and a 1000-way fc layer for ImageNet classification. The average pooling and the fc layers are removed. A randomly initialized $1 \times 1$ convolution is added at last to reduce the channel dimension to 1024. As in common practice [4, 7], the effective stride in the last convolutional block is reduced from 32 pixels to 16 pixels to increase the feature map resolution. Specifically, at the beginning of the last block, stride is changed from 2 to 1 ("conv5" for both ResNet-101 and Aligned-Inception-ResNet). To compensate, the dilation of all the convolution filters in this block (with kernel size $> 1$) is changed from 1 to 2.

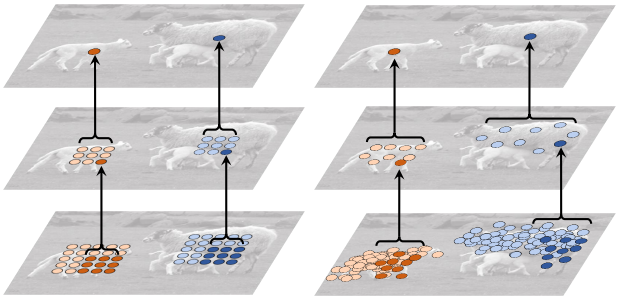
Optionally, *deformable convolution* is applied to the last few convolutional layers (with kernel size $> 1$). We experimented with different numbers of such layers and found 3 as a good trade-off for different tasks, as reported in Table 1.

**Segmentation and Detection Networks** A task specific network is built upon the output feature maps from the feature extraction network mentioned above.

In the below, $C$ denotes the number of object classes.

*DeepLab* [5] is a state-of-the-art method for semantic segmentation. It adds a $1 \times 1$ convolutional layer over the feature maps to generates $(C + 1)$ maps that represent the per-pixel classification scores. A following softmax layer then outputs the per-pixel probabilities.

*Category-Aware RPN* is almost the same as the region proposal network in [47], except that the 2-class (object or not) convolutional classifier is replaced by a $(C + 1)$-class convolutional classifier. It can be considered as a simplified version of SSD [40].



(a) standard convolution (b) deformable convolution

Figure 5: Illustration of the fixed receptive field in standard convolution (a) and the adaptive receptive field in deformable convolution (b), using two layers. Top: two activation units on the top feature map, on two objects of different scales and shapes. The activation is from a $3 \times 3$ filter. Middle: the sampling locations of the $3 \times 3$ filter on the preceding feature map. Another two activation units are highlighted. Bottom: the sampling locations of two levels of $3 \times 3$ filters on the preceding feature map. Two sets of locations are highlighted, corresponding to the highlighted units above.

*Faster R-CNN* [47] is the state-of-the-art detector. In our implementation, the RPN branch is added on the top of the conv4 block, following [47]. In the previous practice [22, 24], the RoI pooling layer is inserted between the conv4 and the conv5 blocks in ResNet-101, leaving 10 layers for each RoI. This design achieves good accuracy but has high per-RoI computation. Instead, we adopt a simplified design as in [38]. The RoI pooling layer is added at last[1]. On top of the pooled RoI features, two fc layers of dimension 1024 are added, followed by the bounding box regression and the classification branches. Although such simplification (from 10 layer conv5 block to 2 fc layers) would slightly decrease the accuracy, it still makes a strong enough baseline and is not a concern in this work.

Optionally, the RoI pooling layer can be changed to *deformable RoI pooling*.

*R-FCN* [7] is another state-of-the-art detector. It has negligible per-RoI computation cost. We follow the original implementation. Optionally, its RoI pooling layer can be changed to *deformable position-sensitive RoI pooling*.

## 3. Understanding Deformable ConvNets

This work is built on the idea of augmenting the spatial sampling locations in convolution and RoI pooling with ad-

[1]The last $1 \times 1$ dimension reduction layer is changed to outputs 256-D features.

Figure 6: Each image triplet shows the sampling locations ($9^3 = 729$ red points in each image) in three levels of $3 \times 3$ deformable filters (see Figure 5 as a reference) for three activation units (green points) on the background (left), a small object (middle), and a large object (right), respectively.

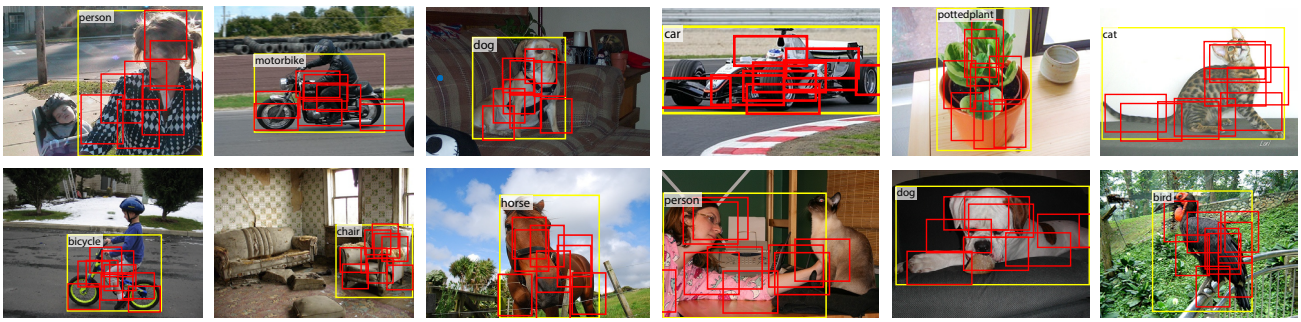



Figure 7: Illustration of offset parts in deformable (positive sensitive) RoI pooling in R-FCN [7] and $3 \times 3$ bins (red) for an input RoI (yellow). Note how the parts are offset to cover the non-rigid objects.

ditional offsets and learning the offsets from target tasks.

When the deformable convolution are stacked, the effect of composited deformation is profound. This is exemplified in Figure 5. The receptive field and the sampling locations in the standard convolution are fixed all over the top feature map (left). They are adaptively adjusted according to the objects' scale and shape in deformable convolution (right). More examples are shown in Figure 6. Table 2 provides quantitative evidence of such adaptive deformation.

The effect of deformable RoI pooling is similar, as illustrated in Figure 7. The regularity of the grid structure in standard RoI pooling no longer holds. Instead, parts deviate from the RoI bins and move onto the nearby object foreground regions. The localization capability is enhanced, especially for non-rigid objects.

### 3.1. In Context of Related Works

Our work is related to previous works in different aspects. We discuss the relations and differences in details.

**Spatial Transform Networks (STN) [26]** It is the first work to learn spatial transformation from data in a deep learning framework. *It warps the feature map via a global parametric transformation* such as affine transformation. Such warping is expensive and learning the transformation parameters is known difficult. STN has shown successes in small scale image classification problems. The inverse STN method [37] replaces the expensive feature warping by efficient transformation parameter propagation.

The offset learning in deformable convolution can be considered as an extremely light-weight spatial transformer in STN [26]. However, *deformable convolution does not adopt a global parametric transformation and feature warping. Instead, it samples the feature map in a local and dense manner.* To generate new feature maps, it has a weighted summation step, which is absent in STN.

Deformable convolution is easy to integrate into any CNN architectures. Its training is easy. It is shown effective for complex vision tasks that require dense (*e.g.*, semantic segmentation) or semi-dense (*e.g.*, object detection) predictions. These tasks are difficult (if not infeasible) for STN [26, 37].

**Active Convolution [27]** This work is contemporary. It also augments the sampling locations in the convolution with offsets and learns the offsets via back-propagation end-to-end. It is shown effective on image classification tasks.

Two crucial differences from deformable convolution

make this work less general and adaptive. First, it shares the offsets all over the different spatial locations. Second, the offsets are *static model parameters* that are learnt per task or per training. In contrast, the offsets in deformable convolution are *dynamic model outputs* that vary per image location. They model the dense spatial transformations in the images and are effective for (semi-)dense prediction tasks such as object detection and semantic segmentation.

**Effective Receptive Field [43]** It finds that not all pixels in a receptive field contribute equally to an output response. The pixels near the center have much larger impact. The effective receptive field only occupies a small fraction of the theoretical receptive field and has a Gaussian distribution. Although the theoretical receptive field size increases linearly with the number of convolutional layers, a surprising result is that, the effective receptive field size increases linearly with the *square root* of the number, therefore, *at a much slower rate* than what we would expect.

This finding indicates that even the top layer's unit in deep CNNs may not have large enough receptive field. This partially explains why atrous convolution [23] is widely used in vision tasks (see below). It indicates the needs of adaptive receptive field learning.

Deformable convolution is capable of learning receptive fields adaptively, as shown in Figure 5, 6 and Table 2.

**Atrous convolution [23]** It increases a normal filter's stride to be larger than 1 and keeps the original weights at sparsified sampling locations. This increases the receptive field size and retains the same complexity in parameters and computation. It has been widely used for semantic segmentation [41, 5, 54] (also called dilated convolution in [54]), object detection [7], and image classification [55].

Deformable convolution is a generalization of atrous convolution, as easily seen in Figure 1 (c). Extensive comparison to atrous convolution is presented in Table 3.

**Deformable Part Models (DPM) [11]** Deformable RoI pooling is similar to DPM because both methods learn the spatial deformation of object parts to maximize the classification score. Deformable RoI pooling is simpler since no spatial relations between the parts are considered.

DPM is a shallow model and has limited capability of modeling deformation. While its inference algorithm can be converted to CNNs [17] by treating the distance transform as a special pooling operation, its training is not end-to-end and involves heuristic choices such as selection of components and part sizes. In contrast, deformable ConvNets are deep and perform end-to-end training. When multiple deformable modules are stacked, the capability of modeling deformation becomes stronger.

**DeepID-Net [44]** It introduces a deformation constrained pooling layer which also considers part deformation for object detection. It therefore shares a similar spirit with deformable RoI pooling, but is much more complex. This work is highly engineered and based on RCNN [16]. It is unclear how to adapt it to the recent state-of-the-art object detection methods [47, 7] in an end-to-end manner.

**Spatial manipulation in RoI pooling** Spatial pyramid pooling [34] uses hand crafted pooling regions over scales. It is the predominant approach in computer vision and also used in deep learning based object detection [21, 15].

Learning the spatial layout of pooling regions has received little study. The work in [28] learns a sparse subset of pooling regions from a large over-complete set. The large set is hand engineered and the learning is not end-to-end.

Deformable RoI pooling is the first to learn pooling regions end-to-end in CNNs. While the regions are of the same size currently, extension to multiple sizes as in spatial pyramid pooling [34] is straightforward.

**Transformation invariant features and their learning** There have been tremendous efforts on designing transformation invariant features. Notable examples include scale invariant feature transform (SIFT) [42] and ORB [49] (O for orientation). There is a large body of such works in the context of CNNs. The invariance and equivalence of CNN representations to image transformations are studied in [36]. Some works learn invariant CNN representations with respect to different types of transformations such as [50], scattering networks [3], convolutional jungles [32], and TI-pooling [33]. Some works are devoted for specific transformations such as symmetry [13, 9], scale [29], and rotation [53].

As analyzed in Section 1, in these works the transformations are known a priori. The knowledge (such as parameterization) is used to hand craft the structure of feature extraction algorithm, either fixed in such as SIFT, or with learnable parameters such as those based on CNNs. They cannot handle unknown transformations in the new tasks.

In contrast, our deformable modules generalize various transformations (see Figure 1). The transformation invariance is learned from the target task.

**Dynamic Filter [2]** Similar to deformable convolution, the dynamic filters are also conditioned on the input features and change over samples. Differently, only the filter weights are learned, not the sampling locations like ours. This work is applied for video and stereo prediction.

**Combination of low level filters** Gaussian filters and its smooth derivatives [30] are widely used to extract low level image structures such as corners, edges, T-junctions, etc. Under certain conditions, such filters form a set of basis and their linear combination forms new filters within the same group of geometric transformations, such as multiple orientations in *Steerable Filters* [12] and multiple scales in [45]. We note that although the term *deformable kernels* is used in [45], its meaning is different from ours in this work.

Most CNNs learn all their convolution filters from scratch. The recent work [25] shows that it could be unnec-

| usage of deformable convolution (# layers) | DeepLab | | class-aware RPN | | Faster R-CNN | | R-FCN | |
|---|---|---|---|---|---|---|---|---|
| | mIoU@V (%) | mIoU@C (%) | mAP@0.5 (%) | mAP@0.7 (%) | mAP@0.5 (%) | mAP@0.7 (%) | mAP@0.5 (%) | mAP@0.7 (%) |
| none (0, baseline) | 69.7 | 70.4 | 68.0 | 44.9 | 78.1 | 62.1 | 80.0 | 61.8 |
| res5c (1) | 73.9 | 73.5 | 73.5 | 54.4 | 78.6 | 63.8 | 80.6 | 63.0 |
| res5b,c (2) | 74.8 | 74.4 | 74.3 | 56.3 | 78.5 | 63.3 | 81.0 | 63.8 |
| res5a,b,c (3, default) | **75.2** | **75.2** | 74.5 | 57.2 | 78.6 | 63.3 | 81.4 | 64.7 |
| res5 & res4b22,b21,b20 (6) | 74.8 | 75.1 | **74.6** | **57.7** | **78.7** | **64.0** | **81.5** | **65.4** |

Table 1: Results of using deformable convolution in the last 1, 2, 3, and 6 convolutional layers (of $3 \times 3$ filter) in ResNet-101 feature extraction network. For *class-aware RPN*, *Faster R-CNN*, and *R-FCN*, we report result on VOC 2007 test.

| layer | small | medium | large | background |
|---|---|---|---|---|
| | mean $\pm$ std | | | |
| res5c | $5.3 \pm 3.3$ | $5.8 \pm 3.5$ | $8.4 \pm 4.5$ | $6.2 \pm 3.0$ |
| res5b | $2.5 \pm 1.3$ | $3.1 \pm 1.5$ | $5.1 \pm 2.5$ | $3.2 \pm 1.2$ |
| res5a | $2.2 \pm 1.2$ | $2.9 \pm 1.3$ | $4.2 \pm 1.6$ | $3.1 \pm 1.1$ |

Table 2: Statistics of effective dilation values of deformable convolutional filters on three layers and four categories. Similar as in COCO [39], we divide the objects into three categories equally according to the bounding box area. Small: area $< 96^2$ pixels; medium: $96^2 <$ area $< 224^2$; large: area $> 224^2$ pixels.

essary. It replaces the free form filters by weighted combination of low level filters (Gaussian derivatives up to 4-th order) and learns the weight coefficients. The regularization over the filter function space is shown to improve the generalization ability when training data are small.

Above works are related to ours in that, when multiple filters, especially with different scales, are combined, the resulting filter could have complex weights and resemble our deformable convolution filter. However, deformable convolution learns sampling locations instead of filter weights.

# 4. Experiments

## 4.1. Experiment Setup and Implementation

**Semantic Segmentation** We use *PASCAL VOC* [10] and *CityScapes* [6]. For *PASCAL VOC*, there are 20 semantic categories. Following the protocols in [19, 41, 4], we use VOC 2012 dataset and the additional mask annotations in [18]. The training set includes $10,582$ images. Evaluation is performed on $1,449$ images in the validation set. For *CityScapes*, following the protocols in [5], training and evaluation are performed on $2,975$ images in the train set and 500 images in the validation set, respectively. There are 19 semantic categories plus a background category.

For evaluation, we use the mean intersection-over-union (mIoU) metric defined over image pixels, following the standard protocols [10, 6]. We use mIoU@V and mIoU@C for PASCAl VOC and Cityscapes, respectively.

In training and inference, the images are resized to have a shorter side of 360 pixels for PASCAL VOC and $1,024$ pixels for Cityscapes. In SGD training, one image is randomly sampled in each mini-batch. A total of 30k and 45k iterations are performed for PASCAL VOC and Cityscapes, respectively, with 8 GPUs and one mini-batch on each. The learning rates are $10^{-3}$ and $10^{-4}$ in the first $\frac{2}{3}$ and the last $\frac{1}{3}$ iterations, respectively.

**Object Detection** We use *PASCAL VOC* and *COCO* [39] datasets. For *PASCAL VOC*, following the protocol in [15], training is performed on the union of VOC 2007 trainval and VOC 2012 trainval. Evaluation is on VOC 2007 test. For *COCO*, following the standard protocol [39], training and evaluation are performed on the 120k images in the trainval and the 20k images in the test-dev, respectively.

For evaluation, we use the standard mean average precision (mAP) scores [10, 39]. For PASCAL VOC, we report mAP scores using IoU thresholds at 0.5 and 0.7. For COCO, we use the standard COCO metric of mAP@[0.5:0.95], as well as mAP@0.5.

In training and inference, the images are resized to have a shorter side of 600 pixels. In SGD training, one image is randomly sampled in each mini-batch. For *class-aware RPN*, 256 RoIs are sampled from the image. For *Faster R-CNN* and *R-FCN*, 256 and 128 RoIs are sampled for the region proposal and the object detection networks, respectively. $7 \times 7$ bins are adopted in RoI pooling. To facilitate the ablation experiments on VOC, we follow [38] and utilize pre-trained and fixed RPN proposals for the training of Faster R-CNN and R-FCN, without feature sharing between the region proposal and the object detection networks. The RPN network is trained separately as in the first stage of the procedure in [47]. For COCO, joint training as in [48] is performed and feature sharing is enabled for training. A total of 30k and 240k iterations are performed for PASCAL VOC and COCO, respectively, on 8 GPUs. The learning rates are set as $10^{-3}$ and $10^{-4}$ in the first $\frac{2}{3}$ and the last $\frac{1}{3}$ iterations, respectively.

| deformation modules | DeepLab<br>mIoU@V / @C | class-aware RPN<br>mAP@0.5 / @0.7 | Faster R-CNN<br>mAP@0.5 / @0.7 | R-FCN<br>mAP@0.5 / @0.7 |
|---|---|---|---|---|
| atrous convolution (2,2,2) (default) | 69.7 / 70.4 | 68.0 / 44.9 | 78.1 / 62.1 | 80.0 / 61.8 |
| atrous convolution (4,4,4) | 73.1 / 71.9 | 72.8 / 53.1 | 78.6 / 63.1 | 80.5 / 63.0 |
| atrous convolution (6,6,6) | 73.6 / 72.7 | 73.6 / 55.2 | 78.5 / 62.3 | 80.2 / 63.5 |
| atrous convolution (8,8,8) | 73.2 / 72.4 | 73.2 / 55.1 | 77.8 / 61.8 | 80.3 / 63.2 |
| deformable convolution | **75.3 / 75.2** | **74.5 / 57.2** | 78.6 / 63.3 | 81.4 / 64.7 |
| deformable RoI pooling | N.A | N.A | 78.3 / 66.6 | 81.2 / 65.0 |
| deformable convolution & RoI pooling | N.A | N.A | **79.3 / 66.9** | **82.6 / 68.5** |

Table 3: Evaluation of our deformable modules and atrous convolution, using ResNet-101.

| method | # params | net. forward (sec) | runtime (sec) |
|---|---|---|---|
| DeepLab@C | 46.0 M | 0.610 | 0.650 |
| **Ours** | 46.1 M | 0.656 | 0.696 |
| DeepLab@V | 46.0 M | 0.084 | 0.094 |
| **Ours** | 46.1 M | 0.088 | 0.098 |
| class-aware RPN | 46.0 M | 0.142 | 0.323 |
| **Ours** | 46.1 M | 0.152 | 0.334 |
| Faster R-CNN | 58.3 M | 0.147 | 0.190 |
| **Ours** | 59.9 M | 0.192 | 0.234 |
| R-FCN | 47.1 M | 0.143 | 0.170 |
| **Ours** | 49.5 M | 0.169 | 0.193 |

Table 4: Model complexity and runtime comparison of deformable ConvNets and the plain counterparts, using ResNet-101. The overall runtime in the last column includes image resizing, network forward, and post-processing (*e.g.*, NMS for object detection). Runtime is counted on a workstation with Intel E5-2650 v2 CPU and Nvidia K40 GPU.

### 4.2. Ablation Study

Extensive ablation studies are performed to validate the efficacy and efficiency of our approach.

**Deformable Convolution** Table 1 evaluates the effect of deformable convolution using ResNet-101 feature extraction network. Accuracy steadily improves when more deformable convolution layers are used, especially for *DeepLab* and *class-aware RPN*. The improvement saturates when using 3 deformable layers for *DeepLab*, and 6 for others. In the remaining experiments, we use 3 in the feature extraction networks.

We empirically observed that the learned offsets in the deformable convolution layers are highly adaptive to the image content, as illustrated in Figure 5 and Figure 6. To better understand the mechanism of deformable convolution, we define a metric called *effective dilation* for a deformable convolution filter. It is the mean of the distances between all adjacent pairs of sampling locations in the filter. It is a rough measure of the receptive field size of the filter.

We apply the R-FCN network with 3 deformable layers (as in Table 1) on VOC 2007 test images. We categorize the deformable convolution filters into four classes: small, medium, large, and background, according to the ground truth bounding box annotation and where the filter center is. Table 2 reports the statistics (mean and std) of the effective dilation values. It clearly shows that: 1) *the receptive field sizes of deformable filters are correlated with object sizes, indicating that the deformation is effectively learned from image content*; 2) *the filter sizes on the background region are between those on medium and large objects, indicating that a relatively large receptive field is necessary for recognizing the background regions*. These observations are consistent in different layers.

The default ResNet-101 model uses atrous convolution with dilation 2 for the last three $3 \times 3$ convolutional layers (see Section 2.3). We further tried dilation values 4, 6, and 8 and reported the results in Table 3. It shows that: 1) *accuracy increases for all tasks when using larger dilation values, indicating that the default networks have too small receptive fields*; 2) *the optimal dilation values vary for different tasks,* e.g., 6 *for DeepLab but* 4 *for Faster R-CNN;* 3) *deformable convolution has the best accuracy*. These observations verify that adaptive learning of filter deformation is effective and necessary.

**Deformable RoI Pooling** It is applicable to *Faster R-CNN* and *R-FCN*. As shown in Table 3, using it alone already produces noticeable performance gains, especially at the strict mAP@0.7 metric. When both deformable convolution and RoI Pooling are used, significant accuracy improvements are obtained.

**Model Complexity and Runtime** Table 4 reports the model complexity and runtime of the proposed deformable ConvNets and their plain versions. Deformable ConvNets only add small overhead over model parameters and computation. This indicates that the significant performance

| method | backbone architecture | M | B | mAP@[0.5:0.95] | mAP$^r$@0.5 | mAP@[0.5:0.95] (small) | mAP@[0.5:0.95] (mid) | mAP@[0.5:0.95] (large) |
|---|---|---|---|---|---|---|---|---|
| class-aware RPN | ResNet-101 | | | 23.2 | 42.6 | 6.9 | 27.1 | 35.1 |
| **Ours** | | | | **25.8** | **45.9** | **7.2** | **28.3** | **40.7** |
| Faster RCNN | ResNet-101 | | | 29.4 | 48.0 | 9.0 | 30.5 | 47.1 |
| **Ours** | | | | **33.1** | **50.3** | **11.6** | **34.9** | **51.2** |
| R-FCN | ResNet-101 | | | 30.8 | 52.6 | 11.8 | 33.9 | 44.8 |
| **Ours** | | | | **34.5** | **55.0** | **14.0** | **37.7** | **50.3** |
| Faster RCNN | Aligned-Inception-ResNet | | | 30.8 | 49.6 | 9.6 | 32.5 | 49.0 |
| **Ours** | | | | **34.1** | **51.1** | **12.2** | **36.5** | **52.4** |
| R-FCN | Aligned-Inception-ResNet | | | 32.9 | 54.5 | 12.5 | 36.3 | 48.3 |
| **Ours** | | | | **36.1** | **56.7** | **14.8** | **39.8** | **52.2** |
| R-FCN | Aligned-Inception-ResNet | ✓ | | 34.5 | 55.0 | 16.8 | 37.3 | 48.3 |
| **Ours** | | ✓ | | 37.1 | 57.3 | 18.8 | 39.7 | 52.3 |
| R-FCN | | ✓ | ✓ | 35.5 | 55.6 | 17.8 | 38.4 | 49.3 |
| **Ours** | | ✓ | ✓ | **37.5** | **58.0** | **19.4** | **40.1** | **52.5** |

Table 5: Object detection results of deformable ConvNets v.s. plain ConvNets on COCO test-dev set. M denotes multi-scale testing, and B denotes iterative bounding box average in the table.

improvement is from the capability of modeling geometric transformations, other than increasing model parameters.

### 4.3. Object Detection on COCO

In Table 5, we perform extensive comparison between the deformable ConvNets and the plain ConvNets for object detection on COCO test-dev set. We first experiment using ResNet-101 model. The deformable versions of class-aware RPN, Faster R-CNN and R-FCN achieve mAP@[0.5:0.95] scores of 25.8%, 33.1%, and 34.5% respectively, which are 11%, 13%, and 12% relatively higher than their plain-ConvNets counterparts respectively. By replacing ResNet-101 by Aligned-Inception-ResNet in Faster R-CNN and R-FCN, their plain-ConvNet baselines both improve thanks to the more powerful feature representations. And the effective performance gains brought by deformable ConvNets also hold. By further testing on multiple image scales (the image shorter side is in [480, 576, 688, 864, 1200, 1400]) and performing iterative bounding box average [14], the mAP@[0.5:0.95] scores are increased to 37.5% for the deformable version of R-FCN. Note that the performance gain of deformable ConvNets is complementary to these bells and whistles.

## 5. Conclusion

This paper presents deformable ConvNets, which is a simple, efficient, deep, and end-to-end solution to model dense spatial transformations. For the first time, we show that it is feasible and effective to learn dense spatial transformation in CNNs for sophisticated vision tasks, such as object detection and semantic segmentation.

**Acknowledgements**

The Aligned-Inception-ResNet model was trained and investigated by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun in unpublished work.

## A. Deformable Convolution/RoI Pooling Back-propagation

In the deformable convolution Eq. (2), the gradient w.r.t. the offset $\Delta\mathbf{p}_n$ is computed as

$$\begin{aligned}\frac{\partial \mathbf{y}(\mathbf{p}_0)}{\partial \Delta\mathbf{p}_n} &= \sum_{\mathbf{p}_n\in\mathcal{R}} \mathbf{w}(\mathbf{p}_n)\cdot\frac{\partial \mathbf{x}(\mathbf{p}_0+\mathbf{p}_n+\Delta\mathbf{p}_n)}{\partial\Delta\mathbf{p}_n}\\ &= \sum_{\mathbf{p}_n\in\mathcal{R}}\left[\mathbf{w}(\mathbf{p}_n)\cdot\sum_{\mathbf{q}}\frac{\partial G(\mathbf{q},\mathbf{p}_0+\mathbf{p}_n+\Delta\mathbf{p}_n)}{\partial\Delta\mathbf{p}_n}\mathbf{x}(\mathbf{q})\right],\end{aligned} \tag{7}$$

where the term $\frac{\partial G(\mathbf{q},\mathbf{p}_0+\mathbf{p}_n+\Delta\mathbf{p}_n)}{\partial\Delta\mathbf{p}_n}$ can be derived from Eq. (4). Note that the offset $\Delta\mathbf{p}_n$ is 2D and we use $\partial\Delta\mathbf{p}_n$ to denote $\partial\Delta p_n^x$ and $\partial\Delta p_n^y$ for simplicity.

Similarly, in the deformable RoI Pooling module, the

| stage | spatial dim. | Aligned-Inception-ResNet |
|---|---|---|
| conv1 | 112×112 | 7×7, 64, stride 2 |
| conv2 | 56×56 | 3×3 max pool, stride 2 |
| | | [ 256-d IRB ] ×3 |
| conv3 | 28×28 | [ 512-d IRB ] ×4 |
| conv4 | 14×14 | [ 1024-d IRB ] ×23 |
| conv5 | 7×7 | [ 2048-d IRB ] ×3 |
| classifier | 1×1 | global average pool, 1000-d fc, softmax |

Table 6: Network architecture of Aligned-Inception-ResNet. The *Inception Residual Block* (IRB) is detailed in Figure 8.

| Network | # params | top-1 err (%) | top-5 err (%) |
|---|---|---|---|
| ResNet-101 | 46.0M | 23.6 | 7.1 |
| Inception-ResNet-v2 | 54.3M | 19.6 | 4.7 |
| Aligned-Inception-ResNet | 64.3M | 22.1 | 6.0 |

Table 7: Comparison of Aligned-Inception-ResNet with ResNet-101 and Inception-ResNet-v2 on ImageNet-1K validation.

gradient w.r.t. the offset $\Delta\mathbf{p}_{ij}$ can be computed by

$$\frac{\partial \mathbf{y}(i,j)}{\partial \Delta\mathbf{p}_{ij}} = \frac{1}{n_{ij}} \sum_{\mathbf{p}\in bin(i,j)} \frac{\partial \mathbf{x}(\mathbf{p}_0+\mathbf{p}+\Delta\mathbf{p}_{ij})}{\partial \Delta\mathbf{p}_{ij}} = \frac{1}{n_{ij}} \sum_{\mathbf{p}\in bin(i,j)} \left[ \sum_{\mathbf{q}} \frac{\partial G(\mathbf{q}, \mathbf{p}_0+\mathbf{p}+\Delta\mathbf{p}_{ij})}{\partial \Delta\mathbf{p}_{ij}} \mathbf{x}(\mathbf{q}) \right]. \tag{8}$$

And the gradient w.r.t. the normalized offsets $\Delta\widehat{\mathbf{p}}_{ij}$ can be easily obtained via computing derivatives in $\Delta\mathbf{p}_{ij} = \gamma \cdot \Delta\widehat{\mathbf{p}}_{ij} \circ (w, h)$.

## B. Details of Aligned-Inception-ResNet

In the original Inception-ResNet [51] architecture, multiple layers of valid convolution/pooling are utilized, which brings feature alignment issues for dense prediction tasks. For a cell on the feature maps close to the output, its projected spatial location on the image is not aligned with the location of its receptive field center. Meanwhile, the task specific networks are usually designed under the alignment assumption. For example, in the prevalent FCNs for semantic segmentation, the features from a cell are leveraged to predict the pixels label at the corresponding projected image location.

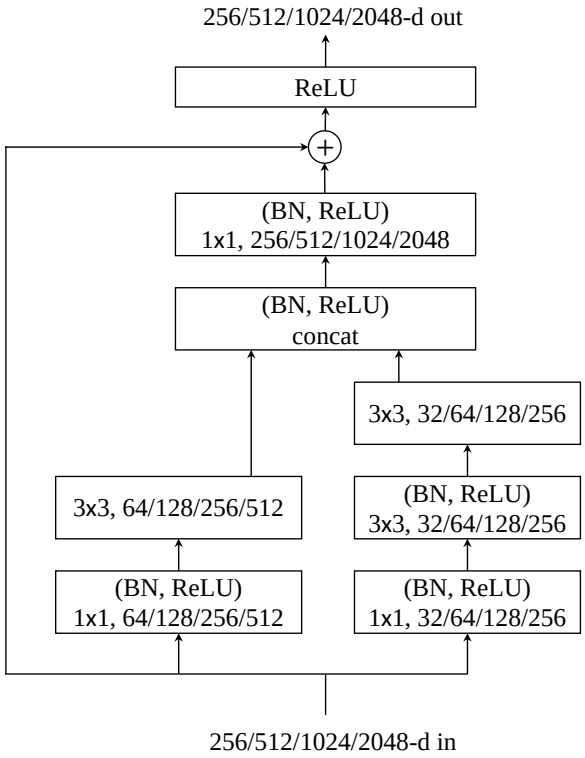



Figure 8: The *Inception Residual Block* (IRB) for different stages of Aligned-Inception-ResNet, where the dimensions of different stages are separated by slash (conv2/conv3/conv4/conv5).

To remedy this issue, the network architecture is modified [20], called "Aligned-Inception-ResNet" and shown in Table 6. When the feature dimension changes, a $1 \times 1$ convolution layer with stride 2 is utilized. There are two main differences between Aligned-Inception-ResNet and the original Inception-ResNet [51]. Firstly, Aligned-Inception-ResNet does not have the feature alignment problem, by proper padding in convolutional and pooling layers. Secondly, Aligned-Inception-ResNet consists of repetitive modules, whose design is simpler than the original Inception-ResNet architectures.

The Aligned-Inception-ResNet model is pre-trained on ImageNet-1K classification [8]. The training procedure follows [22]. Table 7 reports the model complexity, top-1 and

top-5 classification errors.

## References


[1] Y.-L. Boureau, J. Ponce, and Y. LeCun. A theoretical analysis of feature pooling in visual recognition. In *ICML*, 2010. 1

[2] B. D. Brabandere, X. Jia, T. Tuytelaars, and L. V. Gool. Dynamic filter networks. In *NIPS*, 2016. 6

[3] J. Bruna and S. Mallat. Invariant scattering convolution networks. *TPAMI*, 2013. 6

[4] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. In *ICLR*, 2015. 4, 7

[5] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *arXiv preprint arXiv:1606.00915*, 2016. 4, 6, 7

[6] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016. 7

[7] J. Dai, Y. Li, K. He, and J. Sun. R-fcn: Object detection via region-based fully convolutional networks. In *NIPS*, 2016. 1, 2, 3, 4, 5, 6

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 4, 10

[9] S. Dieleman, J. D. Fauw, and K. Kavukcuoglu. Exploiting cyclic symmetry in convolutional neural networks. *arXiv preprint arXiv:1602.02660*, 2016. 6

[10] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes (VOC) Challenge. *IJCV*, 2010. 7

[11] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *TPAMI*, 2010. 2, 6

[12] W. T. Freeman and E. H. Adelson. The design and use of steerable filters. *TPAMI*, 1991. 6

[13] R. Gens and P. M. Domingos. Deep symmetry networks. In *NIPS*, 2014. 6

[14] S. Gidaris and N. Komodakis. Object detection via a multi-region & semantic segmentation-aware cnn model. In *ICCV*, 2015. 9

[15] R. Girshick. Fast R-CNN. In *ICCV*, 2015. 1, 2, 3, 6, 7

[16] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 1, 3, 6

[17] R. Girshick, F. Iandola, T. Darrell, and J. Malik. Deformable part models are convolutional neural networks. *arXiv preprint arXiv:1409.5403*, 2014. 6

[18] B. Hariharan, P. Arbeláez, L. Bourdev, S. Maji, and J. Malik. Semantic contours from inverse detectors. In *ICCV*, 2011. 7

[19] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik. Simultaneous detection and segmentation. In *ECCV*. 2014. 7

[20] K. He, X. Zhang, S. Ren, and J. Sun. Aligned-inception-resnet model, unpublished work. 4, 10

[21] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *ECCV*, 2014. 6

[22] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 4, 10

[23] M. Holschneider, R. Kronland-Martinet, J. Morlet, and P. Tchamitchian. A real-time algorithm for signal analysis with the help of the wavelet transform. *Wavelets: Time-Frequency Methods and Phase Space*, page 289297, 1989. 6

[24] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. *arXiv preprint arXiv:1611.10012*, 2016. 4

[25] J.-H. Jacobsen, J. van Gemert, Z. Lou, and A. W.M.Smeulders. Structured receptive fields in cnns. In *CVPR*, 2016. 6

[26] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial transformer networks. In *NIPS*, 2015. 2, 5

[27] Y. Jeon and J. Kim. Active convolution: Learning the shape of convolution for image classification. In *CVPR*, 2017. 5

[28] Y. Jia, C. Huang, and T. Darrell. Beyond spatial pyramids: Receptive field learning for pooled image features. In *CVPR*, 2012. 6

[29] A. Kanazawa, A. Sharma, and D. Jacobs. Locally scale-invariant convolutional neural networks. In *NIPS*, 2014. 6

[30] J. J. Koenderink and A. J. van Doom. Representation of local geometry in the visual system. *Biological Cybernetics*, 55(6):367–375, Mar. 1987. 6

[31] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 1

[32] D. Laptev and J. M.Buhmann. Transformation-invariant convolutional jungles. In *CVPR*, 2015. 6

[33] D. Laptev, N. Savinov, J. M. Buhmann, and M. Pollefeys. Ti-pooling: transformation-invariant pooling for feature learning in convolutional neural networks. *arXiv preprint arXiv:1604.06318*, 2016. 6

[34] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, 2006. 6

[35] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 1995. 1

[36] K. Lenc and A. Vedaldi. Understanding image representations by measuring their equivariance and equivalence. In *CVPR*, 2015. 6

[37] C.-H. Lin and S. Lucey. Inverse compositional spatial transformer networks. *arXiv preprint arXiv:1612.03897*, 2016. 5

[38] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *CVPR*, 2017. 4, 7

[39] 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 *ECCV*. 2014. 7

[40] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, and S. Reed. Ssd: Single shot multibox detector. In *ECCV*, 2016. 1, 4

[41] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015. 1, 6, 7

[42] D. G. Lowe. Object recognition from local scale-invariant features. In *ICCV*, 1999. 1, 6

[43] W. Luo, Y. Li, R. Urtasun, and R. Zemel. Understanding the effective receptive field in deep convolutional neural networks. *arXiv preprint arXiv:1701.04128*, 2017. 6

[44] W. Ouyang, X. Wang, X. Zeng, S. Qiu, P. Luo, Y. Tian, H. Li, S. Yang, Z. Wang, C.-C. Loy, and X. Tang. Deepid-net: Deformable deep convolutional neural networks for object detection. In *CVPR*, 2015. 6

[45] P. Perona. Deformable kernels for early vision. *TPAMI*, 1995. 6

[46] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, 2016. 1

[47] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*, 2015. 1, 3, 4, 6, 7

[48] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. *TPAMI*, 2016. 7

[49] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: an efficient alternative to sift or surf. In *ICCV*, 2011. 6

[50] K. Sohn and H. Lee. Learning invariant representations with local transformations. In *ICML*, 2012. 6

[51] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261*, 2016. 4, 10

[52] C. Szegedy, S. Reed, D. Erhan, and D. Anguelov. Scalable, high-quality object detection. *arXiv:1412.1441v2*, 2014. 1

[53] D. E. Worrall, S. J. Garbin, D. Turmukhambetov, and G. J. Brostow. Harmonic networks: Deep translation and rotation equivariance. *arXiv preprint arXiv:1612.04642*, 2016. 6

[54] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. In *ICLR*, 2016. 6

[55] F. Yu, V. Koltun, and T. Funkhouser. Dilated residual networks. In *CVPR*, 2017. 6