Deformable Part Models are Convolutional Neural Networks

Ross Girshick[1] Forrest Iandola[2] Trevor Darrell[2] Jitendra Malik[2]
[1]Microsoft Research [2]UC Berkeley
rbg@microsoft.com {forresti,trevor,malik}@eecs.berkeley.edu

Abstract

Deformable part models (DPMs) and convolutional neural networks (CNNs) are two widely used tools for visual recognition. They are typically viewed as distinct approaches: DPMs are graphical models (Markov random fields), while CNNs are "black-box" non-linear classifiers. In this paper, we show that a DPM can be formulated as a CNN, thus providing a synthesis of the two ideas. Our construction involves unrolling the DPM inference algorithm and mapping each step to an equivalent CNN layer. From this perspective, it is natural to replace the standard image features used in DPMs with a learned feature extractor. We call the resulting model a DeepPyramid DPM and experimentally validate it on PASCAL VOC object detection. We find that DeepPyramid DPMs significantly outperform DPMs based on histograms of oriented gradients features (HOG) and slightly outperforms a comparable version of the recently introduced R-CNN detection system, while running significantly faster.

1. Introduction

Part-based representations are widely used in visual recognition. In particular, deformable part models (DPMs) [10] have been effective for generic object category detection. DPMs update pictorial structure models [11], which date back to the 1970s [14], with modern image features and machine learning algorithms.

Convolutional neural networks (CNNs) are another influential class of models for visual recognition. CNNs also have a long history [15, 28, 33], and have resurged over the last two years due to good performance on image classification [27], object detection [17], and more recently a wide variety of vision tasks (*e.g.*, [3, 6, 22, 38, 45]).

These two models, DPMs and CNNs, are typically viewed as distinct approaches to visual recognition. DPMs are graphical models (Markov random fields), while CNNs are "black-box" non-linear classifiers. In this paper, we ask: Are these models actually distinct? To answer this question we show that any DPM can be formulated as an equivalent CNN. In other words, deformable part models *are* convolutional neural networks. Our construction relies on a new network layer, *distance transform pooling*, which generalizes max pooling.

DPMs typically operate on a scale-space pyramid of gradient orientation feature maps (HOG [5]). But we now know that for object detection this feature representation is suboptimal compared to features computed by deep convolutional networks [17]. As a second innovation, we replace HOG with features learned by a fully-convolutional network. This "front-end" network generates a pyramid of deep features, analogous to a HOG feature pyramid. We call the full model a *DeepPyramid DPM*.

We experimentally validate DeepPyramid DPMs by measuring object detection performance on PASCAL VOC [9]. Since traditional DPMs have been tuned for HOG features over many years, we first analyze the differences between HOG feature pyramids and deep feature pyramids. We then select a good model structure and train a DeepPyramid DPM that significantly outperforms the best HOG-based DPMs. While we don't expect our approach to outperform a fine-tuned R-CNN detector [17], we do find that it slightly outperforms a comparable R-CNN (specifically, an R-CNN on the same conv_5 features, without fine-tuning), while running about 20x faster (0.6s *vs*. 12s per image).

Our experiments also shed some light the relative merits of region-based detection methods, such as R-CNN, and sliding-window methods like DPM. We find that region proposals and sliding windows are complementary approaches that will likely benefit each other if used in an ensemble. This makes sense; some object classes are easy to segment (*e.g.*, cats) while others are difficult (*e.g.*, bottles, people).

Interpreted more generally, this paper shows that sliding-window detectors on deep feature pyramids significantly outperform equivalent models on HOG. While not surprising, the implementation details are crucial and challenging to pin down. As a result, HOG-based detectors are still used in a wide range of systems, such as recent hybrid deep/conventional approaches [3, 46], and especially where region-based methods are ill-suited (poselets [1] being a prime example). We therefore believe that the results

presented in this paper will be of broad practical interest to the visual recognition community. An open-source implementation will be made available on the first author's website, which will allow researchers to easily build on our work.

2. DeepPyramid DPMs

A DeepPyramid DPM is a convolutional network that takes as input an image pyramid and produces as output a pyramid of object detection scores. Although the model is a single network, for pedagogical reasons we describe it in terms of two smaller networks, a feature pyramid "front-end" CNN and a DPM-CNN—their function composition yields the full network. A schematic diagram of the model is presented in Figure 1.

2.1. Feature pyramid front-end CNN

Objects appear at all scales in images. A standard technique for coping with this fact is to run a detector at multiple scales using an image pyramid. In the context of CNNs, this method dates back to (at least) early work on face detection in [40], and has been used again in contemporary works, including OverFeat [35], DetectorNet [4], DenseNet [24], and SPP-net [23], a recently proposed method for speeding up R-CNNs. We follow this approach and use as our front-end CNN a network that maps an image pyramid to a feature pyramid. To do this, we use a standard single-scale architecture (Krizhevsky *et al*. [27]) and tie the network weights across all scales. Implementation details are given in Section 3.

2.2. Constructing an equivalent CNN from a DPM

In the DPM formalism, an object class is modeled as a mixture of "components", each being responsible for modeling the appearance of an object sub-category (*e.g*., side views of cars, people doing handstands, bi-wing propeller planes). Each component, in turn, uses a low-resolution global appearance model of the sub-type (called a "root filter"), together with a small number (*e.g*., 8) of higher resolution "part filters" that capture the appearance of local regions of the sub-type.

At test time, a DPM is run as a sliding-window detector over a feature pyramid, which is traditionally built using HOG features (alternatives have recently been explored in [29, 32]). A DPM score is assigned to each sliding-window location by optimizing a score function that trades off part deformation costs with image match scores. A global maximum of the score function is computed efficiently at all sliding-window locations by sharing computation between neighboring positions and using a dynamic programming algorithm. This algorithm is illustrated with all of the steps "unrolled" in Figure 4 of [10]. The key observation of this section is that for any given DPM, its unrolled detection algorithm generates a specific convolutional network architecture of a fixed depth. This architecture, which we call a DPM-CNN, is illustrated for a single-component DPM by the network diagram in Figure 2. In words, the architecture operates in the following way:

1. A DPM-CNN receives a feature pyramid level as input (*e.g*., a conv_5 feature map)
2. It convolves this feature map with the root filter and P part filters, generating $P + 1$ feature maps
3. The P feature maps coming from the part filters are fed into a *distance transform pooling* layer
4. The feature map coming from the root filter is "stacked" (channel-wise concatenated) with the P pooled feature maps from the previous step
5. The resulting $P + 1$ channel feature map is convolved with an *object geometry filter*, which produces the output DPM score map for the input pyramid level

We describe the layers of this network in greater detail in the following subsections. We start by introducing the idea of distance transform (DT) pooling, which generalizes the familiar max-pooling operation used in CNNs. Then, we describe the object geometry layer that encodes the relative offsets of DPM parts. Finally, we describe how to extend a single-component DPM-CNN to a multi-component DPM-CNN using maxout units.

To simplify the presentation, we consider the case where all DPM parts operate at the same resolution as the root filter. Multi-resolution models can be implemented by taking two scales as input and inserting a subsampling layer after each DT-pooling layer.

2.2.1 Distance transform pooling

Here we show that distance transforms of sampled functions [12] generalize max pooling.

First, we define max pooling. Consider a function $f\colon \mathcal{G} \to \mathbb{R}$ defined on a regular grid $\mathcal{G}$. The max pooling operation on f, with a window half-length of k, is also a function $M_f\colon \mathcal{G} \to \mathbb{R}$ that is defined by $M_f(p) = \max_{\Delta p \in \{-k,\ldots,k\}} f(p + \Delta p)$.

Following [12], the distance transform of f is a function $D_f\colon \mathcal{G} \to \mathbb{R}$ defined by $D_f(p) = \max_{q \in \mathcal{G}} (f(q) - d(p - q))$. In the case of DPM, $d(r)$ is as a convex quadratic function $d(r) = ar^2 + br$, where $a > 0$ and b are learnable parameters, which vary from part to part. Intuitively, these parameters define the shape of the distance transform's pooling region.

Max pooling can be expressed equivalently as $M_f(p) = \max_{q \in \mathcal{G}} (f(q) - d_{\text{max}}(p - q))$, where $d_{\text{max}}(r)$ is zero if $r \in \{-k, \ldots, k\}$ and ∞ otherwise. Expressing max pooling as

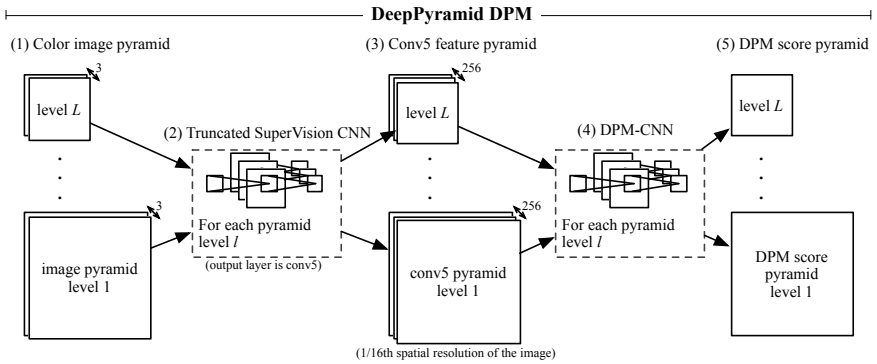

Figure 1. **Schematic model overview.** (1) An image pyramid is built from a color input image. (2) Each pyramid level is forward propagated through a fully-convolutional CNN (*e.g.*, a truncated SuperVision CNN [27] that ends at convolutional layer 5). (3) The result is a pyramid of conv_5 feature maps, each at 1/16th the spatial resolution of its corresponding image pyramid level. (4) Each conv_5 level is then input into a DPM-CNN, which (5) produces a pyramid of DPM detection scores. Since the whole system is the composition of two CNNs, it can be viewed as a single, unified CNN that takes a color image pyramid as input and outputs a DPM score pyramid.

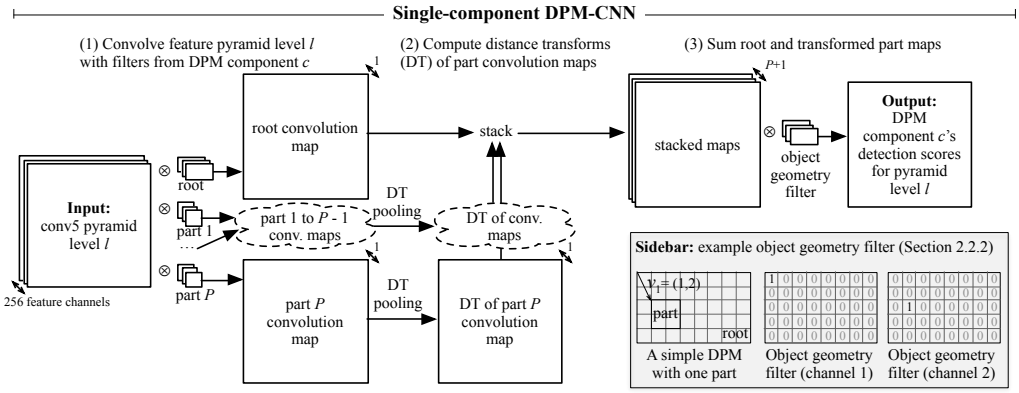

Figure 2. **CNN equivalent to a single-component DPM.** A DPM component can be written as an equivalent CNN by unrolling the DPM detection algorithm into a network. We present the construction for a single-component DPM-CNN here and then show how several of these CNNs can be composed into a multi-component DPM-CNN using a maxout layer (Figure 3). A single-component DPM-CNN operates on a feature pyramid level. (1) The pyramid level is convolved with the root filter and P part filters, yielding $P + 1$ convolution maps. (2) The part convolution maps are then processed with a distance transform pooling layer, which we show is a generalization of max pooling. (3) The root convolution map and the DT pooled part convolution maps are stacked into a single feature map with $P + 1$ channels and then convolved with a sparse object geometry filter (see sidebar diagram and Section 2.2.2). The output is a single-channel score map for the DPM component.

the maximization of a function subject to a distance penalty d_{max} makes the connection between distance transforms and max pooling clear. The distance transform generalizes max pooling and can introduce learnable parameters, as is the case in a DPM. Note that unlike max pooling, the distance transform of f at p is taken over the entire domain $\mathcal{G}$. Therefore, rather than specifying a fixed pooling window a priori, the shape of the pooling region can be learned from the data.

In the construction of a DPM-CNN, DT-pooling layers are inserted after each part filter convolution. When the DT-pooling layer is implemented on a CPU, the distance transform can be computed efficiently in $O(|\mathcal{G}|)$ time, using the algorithm of [12]. When implemented on a GPU, it is faster to loosely bound the pooling region and use a brute-force, but parallel-friendly maximization, as was done in [37].

2.2.2 Object geometry filters

The score of DPM component c at each root filter location s is given by adding the root filter score at s to the distance transformed part scores at "anchor" locations offset from s. Each part p has its own anchor offset that is specified by a 2D vector $v_p = (v_{px}, v_{py})$.

Computing component scores at all root locations can be rephrased as a convolution. The idea is to stack the root filter score map together with the P distance transformed part score maps to form a score map with $P + 1$ channels, and then convolve that score map with a specially constructed

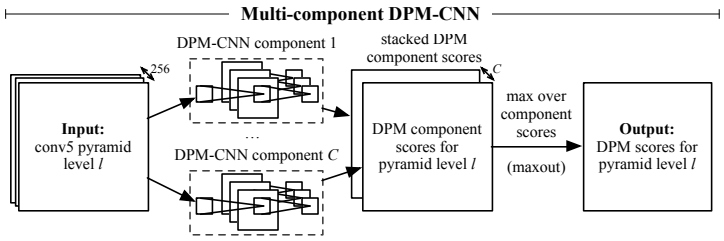

Figure 3. **CNN equivalent to a multi-component DPM.** A multi-component DPM-CNN is composed of one DPM-CNN per component (Figure 2) and a maxout [20] layer that takes a max over component DPM-CNN outputs at each location.

filter that we call an "object geometry" filter. This name comes from the fact that the filter combines a spatial configuration of parts into a whole. The coefficients in the object geometry filter are all zero, except for a single coefficient set to one in each of the filter's $P+1$ channels. The first channel of an object geometry filter always has a one in its upper-left corner (the root's anchor is always defined as $v_0 = (0, 0)$), causing it to "select" the root filter's score. Filter channel p, has a one at index v_p (using matrix-style indexing, where indices grow down and to the right), causing it to select part filter p's score. To clarify this description, an example object geometry filter for a DPM component with one root and one part is shown in the Figure 2 sidebar.

DPMs are usually thought of as flat models, but making the object geometry filter explicit reveals that DPMs actually have a second, implicit convolutional layer. In principle one could train this filter discriminatively, rather than heuristically setting it to a sparse binary pattern [10]. We revisit this idea when discussing our experiments in Section 4.1.

2.2.3 Combining mixture components with maxout

Each single-component DPM-CNN produces a score map for each pyramid level. Let z_{qc} be the score for component c at location q in a pyramid level. In the DPM formalism, components compete with each other at each location. This competition is modeled as a max over component scores: $z_q = \max_c z_{qc}$, where the overall DPM score at q is given by z_q. In DPM-CNN, $z_{qc} = \mathbf{w}_c \cdot \mathbf{x}_q + b_c$, where $\mathbf{w}_c$ is component c's object geometry filter (vectorized), $\mathbf{x}_q$ is the sub-array of root and part scores at q (vectorized), and b_c is the component's scalar bias. Figure 3 shows the full multi-component DPM-CNN including the max non-linearity that combines component scores.

It is interesting to note that the max non-linearity used in DPMs is mathematically equivalent to the "maxout" unit recently described by Goodfellow *et al.* [20]. In the case of DPMs, the maxout unit has a direct interpretation as a switch over the latent choice of model component.

2.2.4 Generalizations

For clarity, we constructed a DPM-CNN for the case of mixtures of star models [10]. However, the construction is general and naturally extends to a variety of models such as object detection grammars [19] and recent DPM variants, such as [44].

The CNN construction for these more complex models is analogous to the DPM-CNN construction: take the exact inference algorithm used for detection and explicitly unroll it (this can be done given a fixed model instance), then express the resulting network in terms of convolutional layers (for appearance and geometry filters), distance transform pooling layers, subsampling layers (if parts are at different scales), and maxout layers (for mixture components).

We also note that our construction is limited to models for which exact inference is possible with a non-iterative algorithm. Models with loopy graphical structures, such as Wang *et al.*'s hierarchical poselets model [43], require iterative, approximate inference algorithms that cannot be converted to equivalent fixed-depth CNNs without further approximation.

2.3. Related work

Our work is most closely related to the deep pedestrian detection model of Ouyang and Wang [31]. Their CNN is structured like a DPM and includes a "deformation layer" that takes a distance penalized global max within a detection window. Their work reveals some connections between single-component DPMs and CNNs, however they focus on a specific model for pedestrian detection and do not present a complete mapping of generic DPMs to CNNs. We extend their work by: (1) developing object geometry filters, (2) showing that multi-component models are implemented with maxout, (3) describing how DPM inference over a whole image is efficiently computed in a CNN by distance transform pooling, and (4) using a more powerful CNN front-end to generate the feature pyramid. Our distance transform pooling layer also differs from their "deformation layer" because it efficiently computes a distance transform over the entire input, rather than an independent global max for each window as described in [31].

Our experimental setup is similar to the contemporaneous work of Savalle *et al*. [34]. Savalle *et al*. show similar results when using deep feature pyramids together with DPMs. Our report complements their work by developing the theoretical relationship between DPMs and CNNs and evaluating more experimental designs.

The idea of unrolling (or "time unfolding") an inference algorithm in order to construct a fixed-depth network was explored by Gregor and LeCun in application to sparse coding [21]. In sparse coding, inference algorithms are iterative and converge to a fixed point. Gregor and LeCun proposed to unroll an inference algorithm for a fixed number of iterations in order to define an approximator network. In the case of DPMs (and similar low tree-width models), the inference algorithm is exact and non-iterative, making it possible to unroll it into a fixed-depth network without any approximations.

Boureau *et al*. [2] study average and max pooling from theoretical and experimental perspectives. They discuss variants of pooling that parametrically transition from average to max. Distance transform pooling, unlike the pooling functions in [2], is interesting because it has a learnable pooling region. Jia *et al*. [25] also address the problem of learning pooling regions by formulating it as a feature selection problem.

Our work is also related to several recent approaches to object detection using CNNs. OverFeat [35] performs coarse sliding-window detection using a CNN. At each rough location, OverFeat uses a regressor to predict the bounding box of a nearby object. Another recent CNN-based object detector called DetectorNet [4] also performs detection on a coarse set of sliding-window locations. At each location, DetectorNet predicts masks for the left, right, top, bottom, and whole object. These predicted masks are then grouped into object hypotheses. Currently, the most accurate object detection method for both ImageNet detection as well as PASCAL VOC is the Region-based CNN framework (R-CNN) [16, 17]. Unlike DetectorNet and OverFeat, R-CNN does not perform sliding-window detection; instead R-CNN begins by extracting a set of region proposals [39] and classifies them with a linear SVM applied to CNN features.

3. Implementation details

We implemented our experiments by modifying the DPM voc-release5 code [18] and using Caffe [26] for CNN computation. We describe the most important implementation details in this section. Source code for the complete system is available, thus providing documentation of the remaining implementation details.

3.1. Parameter learning

There are at least two natural ways to train a DeepPyramid DPM. The first treats the model as a single CNN and trains it end-to-end with SGD and backpropagation. The second trains the model in two stages: (1) fit the front-end CNN; (2) train a DPM on top of stage 1 using latent SVM [10] while keeping the front-end CNN fixed. The first procedure is more in the spirit of deep learning, while the second is an important baseline for comparison to traditional HOG-based DPM and for showing if end-to-end training is useful. We chose to focus on latent SVM training since it is a necessary baseline for end-to-end training. Although we don't explore end-to-end training due to space constraints, we point interested readers to contemporaneous work by Wan *et al*. [41] that shows results of a similar model trained end-to-end. They report modest improvements from end-to-end optimization.

3.2. Feature pyramid construction

Any fully-convolutional network can be used to generate the feature pyramid. In our DeepPyramid DPM implementation, we chose to use a truncated variant of the SuperVision CNN [27]. In order to directly compare our results with R-CNN [17], we use the publicly available network weights that are distributed with R-CNN. These weights were trained on the ILSVRC 2012 classification training dataset using Caffe (we do not use the detection fine-tuned weights since they were trained on warped image windows).

Starting from this network, two structural modifications are required to generate feature pyramids. The first is to truncate it by removing the last max pooling layer (pool_5), all of the fully-connected layers (fc_6, fc_7, fc_8), and the softmax layer. The network's output is, therefore, the feature map computed by the fifth convolutional layer (conv_5), which has 256 feature channels. The second modification is that before each convolutional or max pooling layer, with kernel size k, we zero-pad the layer's input with $\lfloor k/2 \rfloor$ zeros on all sides (top, bottom, left, and right). This padding implements "same" convolution (and pooling), where the input and output maps have the same spatial extent. With this padding, the mapping between image coordinates and CNN output coordinates is straightforward. A "pixel" (or "cell") at zero-based index (x, y) in the CNN's conv_5 feature map has a receptive field centered on pixel $(16x, 16y)$ in the input image. The conv_5 features, therefore, have a stride of 16 pixels in the input image with highly overlapping receptive fields of size 163×163 pixels. Our experimental results show that even though the receptive fields are very large, the features are localized well enough for sliding-window detectors to precisely localize objects without regression (as in OverFeat and DetectorNet). This observation confirms recent observations in [30].

For simplicity, we process the image pyramid with a

naive implementation in which each image pyramid level is embedded in the upper-left corner of a large (1713×1713 pixel) image. For the first image pyramid level, the original image is resized such that its largest dimension is 1713 pixels. For PASCAL VOC images, this results in upsampling images by a factor of 3.4 on average. This upsampling helps compensate for the large 16-pixel stride and facilitates detecting small objects. The first conv_5 pyramid level has 108 cells on its longest side. We use a pyramid with only 7 levels, where the scale factor between levels is $2^{-1/2}$ (the pyramid spans three octaves). The entire conv_5 pyramid has roughly 25k output cells (sliding-window locations). For comparison, this is considerably more than the roughly 1,500 sliding-window locations used in OverFeat, but many fewer than the 250k commonly used in HOG feature pyramids. Computing the conv_5 feature pyramid as described above is fast, even with the naive implementation, taking 0.5 seconds on an NVIDIA Titan Black GPU. This could be made faster by pyramid packing (*e.g.*, [7, 24]).

3.3. DPM training and testing details

Compared to training a DPM with HOG features, we found it necessary to make some changes to the standard DPM training procedure. First, we don't use left/right mirrored pairs of mixture components. These components are easy to implement with HOG because model parameters can be explicitly "flipped" allowing them to be tied between mirrored components. Second, R-CNN and DPM use different non-maximum suppression functions and we found that the one used in R-CNN, which is based on intersection-over-union (IoU) overlap, performs slightly better with conv_5 features (but is worse for the baseline HOG-DPM). Lastly, we found that it's very important to use poorly localized detections of ground-truth objects as negative examples. As in R-CNN, we define negative examples as all detections that have a max IoU overlap with a ground-truth object of less than 0.3. Using poorly localized positives as negative examples leads to substantially better results (Section 4.1.2) than just using negatives from negative images, which is the standard practice when training HOG-DPM. Using these difficult negative examples in HOG-DPM did not improve the baseline results.

4. Experiments

4.1. Exploratory experiments

Deformable part models have been tuned for use with HOG features over several years. A priori, it's unclear if the same structural choices that have worked well for HOG (*e.g.*, the number of mixture components, number of parts, multi-resolution modeling, *etc.*) will also work well with very different features. We conducted several preliminary experiments on the PASCAL VOC 2007 dataset [8] in order to find a DPM structure suited to conv_5 features.

In Table 1, rows 1-3, we show the effect of adding parts to a three component DeepPyramid DPM (three was selected through cross-validation). As in HOG-based DPMs, the dimensions of root filters vary across categories and components, influenced by the aspect ratio distribution for each class. Our root filter sizes typically range from 4×12 to 12×4 feature cells. We start with a "root-only" model (*i.e.*, no parts) and then show results after adding 4 or 8 parts to each component. With 4 parts, mAP increases by 0.9 percentage points, with an improvement in 16 out of 20 classes. The effect size is small, but is statistically significant at $p = 0.016$ under a paired-sample permutation test, a standard method from information retrieval [36].

One significant difference between HOG and conv_5 features is that HOG describes scale-invariant local image statistics (intensity gradients), while conv_5 features describe large (163×163 pixel) image patches. The top two rows of Figure 4 illustrate this point. Each row shows a feature pyramid for an image of a face. The first is HOG and the second is the "face channel" of conv_5. In the HOG representation, the person's face appears at all scales and one can imagine that for each level in the pyramid, it would be possible to define an appropriately sized template that would fire on the face. The conv_5 face channel is quite different. It only shows strong activity when the face is observed in a specific range of scales. In the first several levels of the pyramid, the face feature responses are nearly all zero (black). The feature peaks in level 6 when the face is at the optimal scale.

Based on the previous experiments with parts and the feature visualizations, we decided to explore another hypothesis: that the convolution filters in conv_5 already act as a set of shared "parts" on top of the conv_4 features. This perspective suggests that one can spatially spread the conv_5 features to introduce some local deformation invariance and then learn a root-only DeepPyramid DPM to selectively combine them. This hypothesis is also supported by the features visualized in Figure 4. The heat maps are characteristic of part responses in that they select specific visual structures (cat head, person face, upper-left quarter of a circle) at their locations and scales.

We implemented this idea by applying a 3×3 max filter to conv_5 and then training a root-only DeepPyramid DPM with three components. The max filter is run with a stride of one, instead of the typical used stride of two in max pooling, to prevent subsampling the conv_5 feature map, which would increase the sliding-window stride to 32 pixels. We refer to the max-filtered conv_5 features as "max_5". Note that this model does not have any explicit DPM parts and we can think of the root filters as *learned* object geometry filters that combine the conv_5 "parts". This approach (Table 1 row 4) outperforms the DeepPyramid DPM variants that operate directly on conv_5 in terms of mAP as well as training and

method	C	P	aero	bike	bird	boat	botl	bus	car	cat	chair	cow	table	dog	horse	mbike	pers	plant	sheep	sofa	train	tv	mAP
DP-DPM conv_5	3	0	41.2	64.1	30.5	23.9	35.6	51.8	54.5	37.2	25.8	46.1	38.8	39.1	58.5	54.8	51.6	25.8	**48.3**	33.1	49.1	56.1	43.3
DP-DPM conv_5	3	4	43.1	64.2	31.3	25.0	37.6	55.8	55.7	37.8	27.3	46.0	35.5	39.0	58.2	57.0	53.0	26.6	47.6	35.3	50.6	56.6	44.2
DP-DPM conv_5	3	8	42.3	65.1	32.2	24.4	36.7	56.8	55.7	38.0	**28.2**	47.3	37.1	39.2	61.0	56.4	52.2	26.6	47.0	35.0	51.2	56.1	44.4
DP-DPM max_5	3	0	44.6	65.3	32.7	24.7	35.1	54.3	56.5	40.4	26.3	**49.4**	43.2	**41.0**	61.0	55.7	**53.7**	25.5	47.0	39.8	47.9	**59.2**	45.2
C-DPM [34]	3	8	39.7	59.5	35.8	24.8	35.5	53.7	48.6	46.0	29.2	36.8	45.5	42.0	57.7	56.0	37.4	**30.1**	31.1	**50.4**	56.1	51.6	43.4
Conv-DPM [41]	3	9	48.9	67.3	25.3	25.1	35.7	58.3	60.1	35.3	22.7	36.4	37.1	26.9	64.9	62.0	47.0	24.1	37.5	40.2	54.1	57.0	43.3
Conv-DPM+FT [41]	3	9	50.9	**68.3**	31.9	**28.2**	**38.1**	**61.0**	**61.3**	39.8	25.4	46.5	**47.3**	29.6	**67.5**	**63.4**	46.1	25.2	39.1	45.4	**57.0**	57.9	**46.5**
HOG-DPM	6	0	23.8	51.3	5.1	11.5	19.2	41.3	46.3	8.5	15.8	20.8	8.6	10.4	43.9	37.6	31.9	11.9	18.1	25.7	36.5	35.4	25.2
HOG-DPM [18]	6	8	33.2	60.3	10.2	16.1	27.3	54.3	58.2	23.0	20.0	24.1	26.7	12.7	58.1	48.2	43.2	12.0	21.1	36.1	46.0	43.5	33.7
R-CNN [17] pool_5	n/a	n/a	**51.8**	60.2	**36.4**	27.8	23.2	52.8	60.6	**49.2**	18.3	47.8	44.3	40.8	56.6	58.7	42.4	23.4	46.1	36.7	51.3	55.7	44.2
fine-tuned variants of R-CNN																							
R-CNN FT pool_5	n/a	n/a	58.2	63.3	37.9	27.6	26.1	54.1	66.9	51.4	26.7	55.5	43.4	43.1	57.7	59.0	45.8	28.1	50.8	40.6	53.1	56.4	47.3
R-CNN FT fc_7	n/a	n/a	64.2	69.7	50.0	41.9	32.0	62.6	71.0	60.7	32.7	58.5	46.5	56.1	60.6	66.8	54.2	31.5	52.8	48.9	57.9	64.7	54.2
R-CNN FT fc_7 BB	n/a	n/a	**68.1**	**72.8**	**56.8**	**43.0**	**36.8**	**66.3**	**74.2**	**67.6**	**34.4**	**63.5**	**54.5**	**61.2**	**69.1**	**68.6**	**58.7**	**33.4**	**62.9**	**51.1**	**62.5**	**64.8**	**58.5**

Table 1. **Detection average precision (%) on VOC 2007 test.** Column C shows the number of components and column P shows the number of parts per component. Our method is DP-DPM (DeepPyramid DPM).

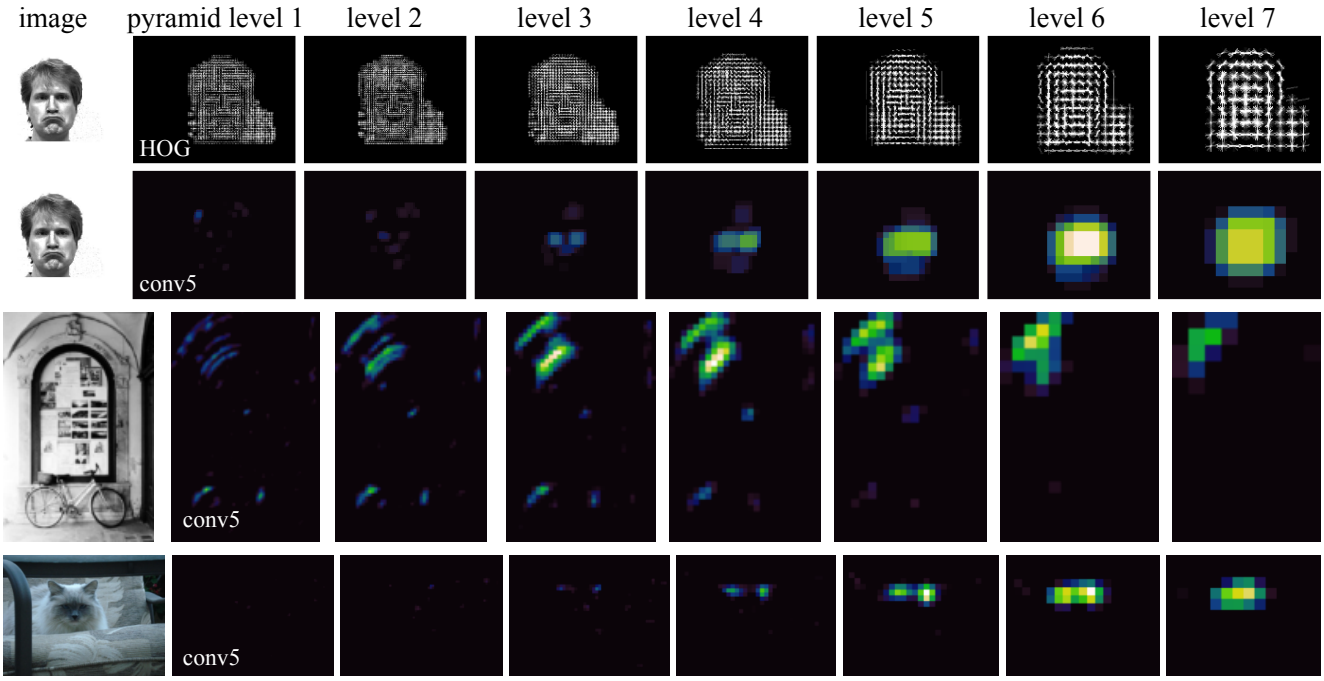

Figure 4. **HOG versus conv_5 feature pyramids.** In contrast to HOG features, conv_5 features are more part-like and scale selective. Each conv_5 pyramid shows 1 of 256 feature channels. The top two rows show a HOG feature pyramid and the "face channel" of a conv_5 pyramid on the same input image. In the HOG pyramid, the face is represented by semi-local gradient features and one can imagine that for each pyramid level, an appropriately sized template could detect the face. The conv_5 face channel, in contrast, is scale selective. The response is almost entirely zero (black) in the first pyramid level and peaks in level 6. Pyramids for two additional images and channels are displayed in rows 3 and 4, demonstrating more scale selectivity. (Face image credit: Yale Face Database.)

testing speed (since each model only has three filters).

4.1.1 Comparison with other methods

We compare our approach to several other methods on the VOC 2007 dataset. The first notable comparison is to DPM on HOG features. We report results using the standard 6 component, 8 part configuration, which achieves a mAP of 33.7%. We also computed another HOG-DPM baseline using 6 components, but without parts. Removing parts decreases HOG-DPM performance to 25.2%. The max_5 variant of DeepPyramid DPM, which uses conv_5 implicitly as

method	aero	bike	bird	boat	botl	bus	car	cat	chair	cow	table	dog	horse	mbike	pers	plant	sheep	sofa	train	tv	mAP
DP-DPM max_5	61.0	55.7	36.5	20.7	33.2	52.5	46.1	48.0	22.1	35.0	32.3	45.7	50.2	59.2	55.8	18.7	49.1	28.8	40.6	48.1	42.0
HOG-DPM [18]	49.2	53.8	13.1	15.3	35.5	53.4	49.7	27.0	17.2	28.8	14.7	17.8	46.4	51.2	47.7	10.8	34.2	20.7	43.8	38.3	33.4
UVA [39]	56.2	42.4	15.3	12.6	21.8	49.3	36.8	46.1	12.9	32.1	30.0	36.5	43.5	52.9	32.9	15.3	41.1	31.8	47.0	44.8	35.1
Regionlets [42]	65.0	48.9	25.9	24.6	24.5	56.1	54.5	51.2	17.0	28.9	30.2	35.8	40.2	55.7	43.5	14.3	43.9	32.6	54.0	45.9	39.7
SegDPM [13]	61.4	53.4	25.6	25.2	35.5	51.7	50.6	50.8	19.3	33.8	26.8	40.4	48.3	54.4	47.1	14.8	38.7	35.0	52.8	43.1	40.4
R-CNN FT fc_7 [17]	67.1	64.1	46.7	32.0	30.5	56.4	57.2	65.9	27.0	47.3	40.9	66.6	57.8	65.9	53.6	26.7	56.5	38.1	52.8	50.2	50.2
R-CNN FT fc_7 BB	**71.8**	**65.8**	**53.0**	**36.8**	**35.9**	**59.7**	**60.0**	**69.9**	**27.9**	**50.6**	**41.4**	**70.0**	**62.0**	**69.0**	**58.1**	**29.5**	**59.4**	**39.3**	**61.2**	**52.4**	**53.7**

Table 2. **Detection average precision (%) on VOC 2010 test.**

a set of shared parts, has a mAP of 45.2%. We also include contemporaneous results from Wan *et al*. [41] and Savalle *et al*. [34]. The baseline method from Wan *et al*. reaches 43.3% mAP, which is then improved to 46.5% by end-to-end fine-tuning.

We also compare our method to the recently proposed R-CNN [17]. The directly comparable version of R-CNN uses $pool_5$ features and no fine-tuning ($pool_5$ is the same as max_5, but with a stride of two instead of one). This comparison isolates differences to the use of a sliding-window method versus classifying warped selective search [39] windows. We can see that for some classes where we expect segmentation to succeed, such as aeroplane and cat, R-CNN strongly outperforms DeepPyramid DPM. For classes where we expect segmentation might under or over segment objects, such as bottle, chair, and person, DeepPyramid DPM strongly outperforms R-CNN. Performance is similar for most of the other categories, with DeepPyramid DPM edging out the $pool_5$ R-CNN in terms of mAP. Of course, this represents the weakest variant of R-CNN, with significant improvements coming from adding fully-connected layers and then fine-tuning for detection and incorporating a bounding-box (BB) regression stage.

We have shown that DeepPyramid DPM is competitive with R-CNN $pool_5$ without fine-tuning. The R-CNN results suggest that most of the gains from fine-tuning come in through the non-linear classifier (implemented via layers fc_6 and fc_7) applied to $pool_5$ features. This suggests that it might be possible to achieve similar levels of performance with DeepPyramid DPM through the use of a more powerful non-linear classifier, although then the model would deviate more strongly from the DPM family.

4.1.2 Ablation studies

To understand the effects of some of our design choices, we report mAP performance on VOC 2007 test using a few ablations of the DP-DPM max_5 model. First, we look at mAP versus the number of mixture components. Mean AP with $\{1, 2, 3\}$ components is $\{39.9\%, 45.1\%, 45.2\%\}$. For most classes, performance improves when going from 1 to 2 or 1 to 3 components because the variety of templates allows for more recall. We also looked at the effect of training with negative examples that come only from negative images (*i.e*., not using mislocalized positives as negative examples), as is done in HOG-DPM. Using negatives only from negative images decreases mAP by 6.3 percentage points to 38.8%. This training strategy, however, does not improve results for HOG-DPM. We also benchmarked the effects of changing non-maximum suppression. Using standard HOG-DPM NMS decreases mAP by 1.3 percentage points, while using the R-CNN variant of NMS does not improve results for HOG-DPM.

4.2. Results on PASCAL VOC 2010-2012

We used the VOC 2007 dataset for model and hyperparameter selection, and now we report results on VOC 2010-2012 obtained using the official evaluation server. Table 2 compares a DeepPyramid DPM with a variety of methods on VOC 2010. The DeepPyramid DPM outperforms all recent methods other than the fine-tuned versions of R-CNN. Performance against HOG-DPM is especially strong. When comparing to R-CNN FT fc_7, without bounding-box regression (BB), DeepPyramid DPM manages better performance in two classes: bottle and person. This likely speaks to the weakness in the region proposals for those classes. The VOC 2011 and 2012 sets are the same and performance is similar to 2010, with a mAP of 41.6%.

5. Conclusion

We have presented a synthesis of deformable part models and convolutional neural networks. This paper demonstrates that any DPM can be expressed as an equivalent CNN by using distance transform pooling, object geometry filters, and maxout units. Distance transform pooling generalizes max pooling and relates the idea of deformable parts to max pooling. We also showed that a DPM-CNN can run on top a feature pyramid constructed by another CNN. The resulting model—which we call a *DeepPyramid DPM*—is a single CNN that performs multi-scale object detection by mapping an image pyramid to a detection score pyramid.

Acknowledgments The GPUs used in this research were generously donated by the NVIDIA Corporation. This work was supported in part by ONR MURI N000141010933, DARPA's MSEE and SMISC programs, NSF awards IIS-1427425 and IIS-1212798, and the Berkeley Vision and Learning Center. R. Girshick was with UC Berkeley while this work was conducted.

References

[1] L. Bourdev and J. Malik. Poselets: Body part detectors trained using 3d human pose annotations. In *ICCV*, 2009.

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

[3] S. Branson, G. V. Horn, S. Belongie, and P. Perona. Bird species categorization using pose normalized deep convolutional nets. In *BMVC*, 2014.

[4] D. E. Christian Szegedy, Alexander Toshev. Deep neural networks for object detection. In *NIPS*, 2013.

[5] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005.

[6] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition. In *ICML*, 2014.

[7] C. Dubout and F. Fleuret. Exact acceleration of linear object detectors. In *ECCV*, 2012.

[8] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

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

[10] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. *TPAMI*, 2010.

[11] P. Felzenszwalb and D. Huttenlocher. Pictorial structures for object recognition. *IJCV*, 2005.

[12] P. Felzenszwalb and D. Huttenlocher. Distance transforms of sampled functions. *Theory of Computing*, 2012.

[13] S. Fidler, R. Mottaghi, A. Yuille, and R. Urtasun. Bottom-up segmentation for top-down detection. In *CVPR*, 2013.

[14] M. Fischler and R. Elschlager. The representation and matching of pictorial structures. *IEEE Trans. on Computer*, 1973.

[15] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 1980.

[16] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

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

[18] R. Girshick, P. Felzenszwalb, and D. McAllester. Discriminatively trained deformable part models, release 5. http://www.cs.berkeley.edu/~rbg/latent-v5/.

[19] R. Girshick, P. Felzenszwalb, and D. McAllester. Object detection with grammar models. In *NIPS*, 2011.

[20] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. In *ICML*, 2013.

[21] K. Gregor and Y. LeCun. Learning fast approximations of sparse coding. In *ICML*, 2010.

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

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

[24] F. Iandola, M. Moskewicz, S. Karayev, R. Girshick, T. Darrell, and K. Keutzer. Densenet: Implementing efficient convnet descriptor pyramids. *arXiv preprint arXiv:1404.1869*, 2014.

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

[26] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proc. of the ACM International Conf. on Multimedia*, 2014.

[27] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, 2012.

[28] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Comp.*, 1989.

[29] J. J. Lim, C. L. Zitnick, and P. Dollár. Sketch tokens: A learned mid-level representation for contour and object detection. In *CVPR*, 2013.

[30] J. Long, N. Zhang, and T. Darrell. Do convnets learn correspondence? In *NIPS (to appear)*, 2014.

[31] W. Ouyang and X. Wang. Joint deep learning for pedestrian detection. In *ICCV*, 2013.

[32] X. Ren and D. Ramanan. Histograms of sparse codes for object detection. In *CVPR*, 2013.

[33] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. *Parallel Distributed Processing*, 1:318–362, 1986.

[34] P.-A. Savalle, S. Tsogkas, G. Papandreou, and I. Kokkinos. Deformable part models with CNN features. In *3rd Parts and Attributes Workshop, ECCV*, 2014.

[35] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. In *ICLR*, 2014.

[36] M. D. Smucker, J. Allan, and B. Carterette. A comparison of statistical significance tests for information retrieval evaluation. In *Conf. on Information and Knowledge Management*, 2007.

[37] H. Song, S. Zickler, T. Althoff, R. Girshick, M. Fritz, C. Geyer, P. Felzenszwalb, and T. Darrell. Sparselet models for efficient multiclass object detection. In *ECCV*, 2012.

[38] J. Tompson, A. Jain, Y. LeCun, and C. Bregler. Joint training of a convolutional network and a graphical model for human pose estimation. In *NIPS (to appear)*, 2014.

[39] J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. Selective search for object recognition. *IJCV*, 2013.

[40] R. Vaillant, C. Monrocq, and Y. LeCun. Original approach for the localisation of objects in images. *IEE Proc. on Vision, Image, and Signal Processing*, 1994.

[41] L. Wan, D. Eigen, and R. Fergus. End-to-end integration of a convolutional network, deformable parts model and non-maximum suppression. *arXiv e-prints*, arXiv:1411.5309 [cs.CV].

[42] X. Wang, M. Yang, S. Zhu, and Y. Lin. Regionlets for generic object detection. In *ICCV*, 2013.

[43] Y. Wang, D. Tran, and Z. Liao. Learning hierarchical poselets for human parsing. In *CVPR*. IEEE, 2011.

[44] Y. Yang and D. Ramanan. Articulated human detection with flexible mixtures-of-parts. *TPAMI*, 2012.

[45] N. Zhang, J. Donahue, R. Girshick, and T. Darrell. Part-based R-CNNs for fine-grained category detection. In *ECCV*, 2014.

[46] N. Zhang, M. Paluri, M. Ranzato, T. Darrell, and L. Bourdev. PANDA: Pose aligned networks for deep attribute modeling. In *CVPR*, 2014.