This CVPR paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the version available on IEEE Xplore.

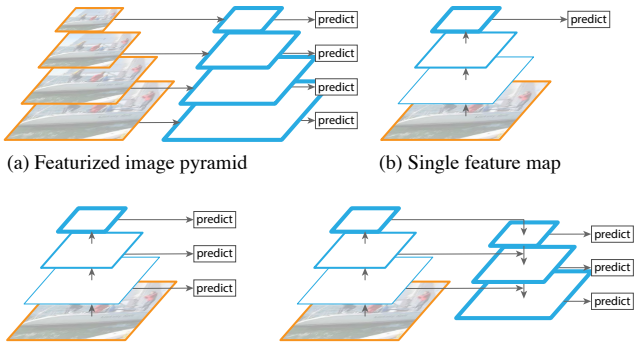
Feature Pyramid Networks for Object Detection

Tsung-Yi Lin[1,2], Piotr Dollár[1], Ross Girshick[1],
Kaiming He[1], Bharath Hariharan[1], and Serge Belongie[2]

[1]Facebook AI Research (FAIR)
[2]Cornell University and Cornell Tech

Abstract

Feature pyramids are a basic component in recognition systems for detecting objects at different scales. But recent deep learning object detectors have avoided pyramid representations, in part because they are compute and memory intensive. In this paper, we exploit the inherent multi-scale, pyramidal hierarchy of deep convolutional networks to construct feature pyramids with marginal extra cost. A top-down architecture with lateral connections is developed for building high-level semantic feature maps at all scales. This architecture, called a Feature Pyramid Network (FPN), shows significant improvement as a generic feature extractor in several applications. Using FPN in a basic Faster R-CNN system, our method achieves state-of-the-art single-model results on the COCO detection benchmark without bells and whistles, surpassing all existing single-model entries including those from the COCO 2016 challenge winners. In addition, our method can run at 5 FPS on a GPU and thus is a practical and accurate solution to multi-scale object detection. Code will be made publicly available.



(a) Featurized image pyramid (b) Single feature map

(c) Pyramidal feature hierarchy (d) Feature Pyramid Network

Figure 1. (a) Using an image pyramid to build a feature pyramid. Features are computed on each of the image scales independently, which is slow. (b) Recent detection systems have opted to use only single scale features for faster detection. (c) An alternative is to reuse the pyramidal feature hierarchy computed by a ConvNet as if it were a featurized image pyramid. (d) Our proposed Feature Pyramid Network (FPN) is fast like (b) and (c), but more accurate. In this figure, feature maps are indicate by blue outlines and thicker outlines denote semantically stronger features.

1. Introduction

Recognizing objects at vastly different scales is a fundamental challenge in computer vision. *Feature pyramids built upon image pyramids* (for short we call these *featurized image pyramids*) form the basis of a standard solution [1] (Fig. 1(a)). These pyramids are scale-invariant in the sense that an object's scale change is offset by shifting its level in the pyramid. Intuitively, this property enables a model to detect objects across a large range of scales by scanning the model over both positions and pyramid levels.

Featurized image pyramids were heavily used in the era of hand-engineered features [5, 25]. They were so critical that object detectors like DPM [7] required dense scale sampling to achieve good results (*e.g.*, 10 scales per octave). For recognition tasks, engineered features have largely been replaced with features computed by deep convolutional networks (ConvNets) [19, 20]. Aside from being capable of representing higher-level semantics, ConvNets are also more robust to variance in scale and thus facilitate recognition from features computed on a single input scale [15, 11, 29] (Fig. 1(b)). But even with this robustness, pyramids are still needed to get the most accurate results. All recent top entries in the ImageNet [33] and COCO [21] detection challenges use multi-scale testing on featurized image pyramids (*e.g.*, [16, 35]). The principle advantage of featurizing each level of an image pyramid is that it produces a multi-scale feature representation in which *all levels are semantically strong*, including the high-resolution levels.

Nevertheless, featurizing each level of an image pyramid has obvious limitations. Inference time increases considerably (*e.g.*, by four times [11]), making this approach impractical for real applications. Moreover, training deep

networks end-to-end on an image pyramid is infeasible in terms of memory, and so, if exploited, image pyramids are used only at test time [15, 11, 16, 35], which creates an inconsistency between train/test-time inference. For these reasons, Fast and Faster R-CNN [11, 29] opt to not use featurized image pyramids under default settings.

However, image pyramids are not the only way to compute a multi-scale feature representation. A deep ConvNet computes a *feature hierarchy* layer by layer, and with subsampling layers the feature hierarchy has an inherent multi-scale, pyramidal shape. This in-network feature hierarchy produces feature maps of different spatial resolutions, but introduces large semantic gaps caused by different depths. The high-resolution maps have low-level features that harm their representational capacity for object recognition.

The Single Shot Detector (SSD) [22] is one of the first attempts at using a ConvNet's pyramidal feature hierarchy as if it were a featurized image pyramid (Fig. 1(c)). Ideally, the SSD-style pyramid would reuse the multi-scale feature maps from different layers computed in the forward pass and thus come free of cost. But to avoid using low-level features SSD foregoes reusing already computed layers and instead builds the pyramid starting from high up in the network (*e.g.*, conv4_3 of VGG nets [36]) and then by adding several new layers. Thus it misses the opportunity to reuse the higher-resolution maps of the feature hierarchy. We show that these are important for detecting small objects.

The goal of this paper is to naturally leverage the pyramidal shape of a ConvNet's feature hierarchy while creating a feature pyramid that has strong semantics at all scales. To achieve this goal, we rely on an architecture that combines low-resolution, semantically strong features with high-resolution, semantically weak features via a top-down pathway and lateral connections (Fig. 1(d)). The result is a feature pyramid that has rich semantics at all levels and is built quickly from a single input image scale. In other words, we show how to create in-network feature pyramids that can be used to replace featurized image pyramids without sacrificing representational power, speed, or memory.

Similar architectures adopting top-down and skip connections are popular in recent research [28, 17, 8, 26]. Their goals are to produce a single high-level feature map of a fine resolution on which the predictions are to be made (Fig. 2 top). On the contrary, our method leverages the architecture as a feature pyramid where predictions (*e.g.*, object detections) are independently made on each level (Fig. 2 bottom). Our model echoes a featurized image pyramid, which has not been explored in these works.

We evaluate our method, called a Feature Pyramid Network (FPN), in various systems for detection and segmentation [11, 29, 27]. Without bells and whistles, we report a state-of-the-art single-model result on the challenging COCO detection benchmark [21] simply based on FPN and a basic Faster R-CNN detector [29], surpassing all existing heavily-engineered single-model entries of competition winners. In ablation experiments, we find that for bounding box proposals, FPN significantly increases the Average Recall (AR) by 8.0 points; for object detection, it improves the COCO-style Average Precision (AP) by 2.3 points and PASCAL-style AP by 3.8 points, over a strong single-scale baseline of Faster R-CNN on ResNets [16]. Our method is also easily extended to mask proposals and improves both instance segmentation AR and speed over state-of-the-art methods that heavily depend on image pyramids.

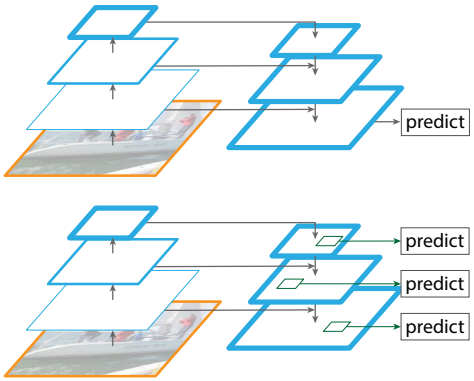


Figure 2. Top: a top-down architecture with skip connections, where predictions are made on the finest level (*e.g.*, [28]). Bottom: our model that has a similar structure but leverages it as a *feature pyramid*, with predictions made independently at all levels.

In addition, our pyramid structure can be trained end-to-end with all scales and is used consistently at train/test time, which would be memory-infeasible using image pyramids. As a result, FPNs are able to achieve higher accuracy than all existing state-of-the-art methods. Moreover, this improvement is achieved without increasing testing time over the single-scale baseline. We believe these advances will facilitate future research and applications. Our code will be made publicly available.

2. Related Work

Hand-engineered features and early neural networks. SIFT features [25] were originally extracted at scale-space extrema and used for feature point matching. HOG features [5], and later SIFT features as well, were computed densely over entire image pyramids. These HOG and SIFT pyramids have been used in numerous works for image classification, object detection, human pose estimation, and more. There has also been significant interest in computing featurized image pyramids quickly. Dollár *et al.* [6] demonstrated fast pyramid computation by first computing a sparsely sampled (in scale) pyramid and then interpolating missing levels. Before HOG and SIFT, early work on face detection with ConvNets [38, 32] computed shallow networks over image pyramids to detect faces across scales.

Deep ConvNet object detectors. With the development of modern deep ConvNets [19], object detectors like OverFeat [34] and R-CNN [12] showed dramatic improvements in accuracy. OverFeat adopted a strategy similar to early neural network face detectors by applying a ConvNet as a sliding window detector on an image pyramid. R-CNN adopted a region proposal-based strategy [37] in which each proposal was scale-normalized before classifying with a ConvNet. SPPnet [15] demonstrated that such region-based detectors could be applied much more efficiently on feature maps extracted on a single image scale. Recent and more accurate detection methods like Fast R-CNN [11] and Faster R-CNN [29] advocate using features computed from a single scale, because it offers a good trade-off between accuracy and speed. Multi-scale detection, however, still performs better, especially for small objects.

Methods using multiple layers. A number of recent approaches improve detection and segmentation by using different layers in a ConvNet. FCN [24] sums partial scores for each category over multiple scales to compute semantic segmentations. Hypercolumns [13] uses a similar method for object instance segmentation. Several other approaches (HyperNet [18], ParseNet [23], and ION [2]) concatenate features of multiple layers before computing predictions, which is equivalent to summing transformed features. SSD [22] and MS-CNN [3] predict objects at multiple layers of the feature hierarchy without combining features or scores.

There are recent methods exploiting lateral/skip connections that associate low-level feature maps across resolutions and semantic levels, including U-Net [31] and SharpMask [28] for segmentation, Recombinator networks [17] for face detection, and Stacked Hourglass networks [26] for keypoint estimation. Ghiasi *et al*. [8] present a Laplacian pyramid presentation for FCNs to progressively refine segmentation. Although these methods adopt architectures with pyramidal shapes, they are unlike featurized image pyramids [5, 7, 34] where predictions are made independently at all levels, see Fig. 2. In fact, for the pyramidal architecture in Fig. 2 (top), image pyramids are still needed to recognize objects across multiple scales [28].

3. Feature Pyramid Networks

Our goal is to leverage a ConvNet's pyramidal feature hierarchy, which has semantics from low to high levels, and build a feature pyramid with high-level semantics throughout. The resulting *Feature Pyramid Network* is general-purpose and in this paper we focus on sliding window proposers (Region Proposal Network, RPN for short) [29] and region-based detectors (Fast R-CNN) [11]. We also generalize FPNs to instance segmentation proposals in Sec. 6.

Our method takes a single-scale image of an arbitrary size as input, and outputs proportionally sized feature maps

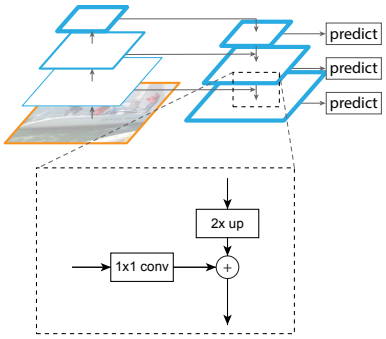


Figure 3. A building block illustrating the lateral connection and the top-down pathway, merged by addition.

at multiple levels, in a fully convolutional fashion. This process is independent of the backbone convolutional architectures (*e.g*., [19, 36, 16]), and in this paper we present results using ResNets [16]. The construction of our pyramid involves a bottom-up pathway, a top-down pathway, and lateral connections, as introduced in the following.

Bottom-up pathway. The bottom-up pathway is the feedforward computation of the backbone ConvNet, which computes a feature hierarchy consisting of feature maps at several scales with a scaling step of 2. There are often many layers producing output maps of the same size and we say these layers are in the same network *stage*. For our feature pyramid, we define one pyramid level for each stage. We choose the output of the last layer of each stage as our reference set of feature maps, which we will enrich to create our pyramid. This choice is natural since the deepest layer of each stage should have the strongest features.

Specifically, for ResNets [16] we use the feature activations output by each stage's last residual block. We denote the output of these last residual blocks as $\{C_2, C_3, C_4, C_5\}$ for conv2, conv3, conv4, and conv5 outputs, and note that they have strides of {4, 8, 16, 32} pixels with respect to the input image. We do not include conv1 into the pyramid due to its large memory footprint.

Top-down pathway and lateral connections. The top-down pathway hallucinates higher resolution features by upsampling spatially coarser, but semantically stronger, feature maps from higher pyramid levels. These features are then enhanced with features from the bottom-up pathway via lateral connections. Each lateral connection merges feature maps of the same spatial size from the bottom-up pathway and the top-down pathway. The bottom-up feature map is of lower-level semantics, but its activations are more accurately localized as it was subsampled fewer times.

Fig. 3 shows the building block that constructs our top-down feature maps. With a coarser-resolution feature map, we upsample the spatial resolution by a factor of 2 (using nearest neighbor upsampling for simplicity). The upsam-

pled map is then merged with the corresponding bottom-up map (which undergoes a 1×1 convolutional layer to reduce channel dimensions) by element-wise addition. This process is iterated until the finest resolution map is generated. To start the iteration, we simply attach a 1×1 convolutional layer on C_5 to produce the coarsest resolution map. Finally, we append a 3×3 convolution on each merged map to generate the final feature map, which is to reduce the aliasing effect of upsampling. This final set of feature maps is called $\{P_2, P_3, P_4, P_5\}$, corresponding to $\{C_2, C_3, C_4, C_5\}$ that are respectively of the same spatial sizes.

Because all levels of the pyramid use shared classifiers/regressors as in a traditional featurized image pyramid, we fix the feature dimension (numbers of channels, denoted as d) in all the feature maps. We set $d = 256$ in this paper and thus all extra convolutional layers have 256-channel outputs. There are no non-linearities in these extra layers, which we have empirically found to have minor impacts.

Simplicity is central to our design and we have found that our model is robust to many design choices. We have experimented with more sophisticated blocks (*e.g.*, using multi-layer residual blocks [16] as the connections) and observed marginally better results. Designing better connection modules is not the focus of this paper, so we opt for the simple design described above.

4. Applications

Our method is a generic solution for building feature pyramids inside deep ConvNets. In the following we adopt our method in RPN [29] for bounding box proposal generation and in Fast R-CNN [11] for object detection. To demonstrate the simplicity and effectiveness of our method, we make minimal modifications to the original systems of [29, 11] when adapting them to our feature pyramid.

4.1. Feature Pyramid Networks for RPN

RPN [29] is a sliding-window class-agnostic object detector. In the original RPN design, a small subnetwork is evaluated on dense 3×3 sliding windows, on top of a single-scale convolutional feature map, performing object/non-object binary classification and bounding box regression. This is realized by a 3×3 convolutional layer followed by two sibling 1×1 convolutions for classification and regression, which we refer to as a network *head*. The object/non-object criterion and bounding box regression target are defined with respect to a set of reference boxes called *anchors* [29]. The anchors are of multiple pre-defined scales and aspect ratios in order to cover objects of different shapes.

We adapt RPN by replacing the single-scale feature map with our FPN. We attach a head of the same design (3×3 conv and two sibling 1×1 convs) to each level on our feature pyramid. Because the head slides densely over all locations in all pyramid levels, it is not necessary to have multi-scale anchors on a specific level. Instead, we assign anchors of a single scale to each level. Formally, we define the anchors to have areas of $\{32^2, 64^2, 128^2, 256^2, 512^2\}$ pixels on $\{P_2, P_3, P_4, P_5, P_6\}$ respectively.[1] As in [29] we also use anchors of multiple aspect ratios $\{1{:}2,\ 1{:}1,\ 2{:}1\}$ at each level. So in total there are 15 anchors over the pyramid.

We assign training labels to the anchors based on their Intersection-over-Union (IoU) ratios with ground-truth bounding boxes as in [29]. Formally, an anchor is assigned a positive label if it has the highest IoU for a given ground-truth box or an IoU over 0.7 with any ground-truth box, and a negative label if it has IoU lower than 0.3 for all ground-truth boxes. Note that scales of ground-truth boxes are not explicitly used to assign them to the levels of the pyramid; instead, ground-truth boxes are associated with anchors, which have been assigned to pyramid levels. As such, we introduce no extra rules in addition to those in [29].

We note that the parameters of the heads are shared across all feature pyramid levels; we have also evaluated the alternative without sharing parameters and observed similar accuracy. The good performance of sharing parameters indicates that all levels of our pyramid share similar semantic levels. This advantage is analogous to that of using a featurized image pyramid, where a common head classifier can be applied to features computed at any image scale.

With the above adaptations, RPN can be naturally trained and tested with our FPN, in the same fashion as in [29]. We elaborate on the implementation details in the experiments.

4.2. Feature Pyramid Networks for Fast R-CNN

Fast R-CNN [11] is a region-based object detector in which Region-of-Interest (RoI) pooling is used to extract features. Fast R-CNN is most commonly performed on a single-scale feature map. To use it with our FPN, we need to assign RoIs of different scales to the pyramid levels.

We view our feature pyramid as if it were produced from an image pyramid. Thus we can adapt the assignment strategy of region-based detectors [15, 11] in the case when they are run on image pyramids. Formally, we assign an RoI of width w and height h (on the input image to the network) to the level P_k of our feature pyramid by:

$$k = \lfloor k_0 + \log_2(\sqrt{wh}/224) \rfloor. \tag{1}$$

Here 224 is the canonical ImageNet pre-training size, and k_0 is the target level on which an RoI with $w \times h = 224^2$ should be mapped into. Analogous to the ResNet-based Faster R-CNN system [16] that uses C_4 as the single-scale feature map, we set k_0 to 4. Intuitively, Eqn. (1) means that if the RoI's scale becomes smaller (say, 1/2 of 224), it should be mapped into a finer-resolution level (say, $k = 3$).

[1] Here we introduce P_6 only for covering a larger anchor scale of 512^2. P_6 is simply a stride two subsampling of P_5. P_6 is not used by the Fast R-CNN detector in the next section.

We attach predictor heads (in Fast R-CNN the heads are class-specific classifiers and bounding box regressors) to all RoIs of all levels. Again, the heads all share parameters, regardless of their levels. In [16], a ResNet's conv5 layers (a 9-layer deep subnetwork) are adopted as the head on top of the conv4 features, but our method has already harnessed conv5 to construct the feature pyramid. So unlike [16], we simply adopt RoI pooling to extract 7×7 features, and attach two hidden 1,024-d fully-connected (*fc*) layers (each followed by ReLU) before the final classification and bounding box regression layers. These layers are randomly initialized, as there are no pre-trained *fc* layers available in ResNets. Note that compared to the standard conv5 head, our 2-*fc* MLP head is lighter weight and faster.

Based on these adaptations, we can train and test Fast R-CNN on top of the feature pyramid. Implementation details are given in the experimental section.

5. Experiments on Object Detection

We perform experiments on the 80 category COCO detection dataset [21]. We train using the union of 80k train images and a 35k subset of val images (`trainval35k` [2]), and report ablations on a 5k subset of val images (`minival`). We also report final results on the standard test set (`test-std`) [21] which has no disclosed labels.

As is common practice [12], all network backbones are pre-trained on the ImageNet1k classification set [33] and then fine-tuned on the detection dataset. We use the pre-trained ResNet-50 and ResNet-101 models that are publicly available.[2] Our code is a reimplementation of `py-faster-rcnn`[3] using Caffe2.[4]

5.1. Region Proposal with RPN

We evaluate the COCO-style Average Recall (AR) and AR on small, medium, and large objects (AR_s, AR_m, and AR_l) following the definitions in [21]. We report results for 100 and 1000 proposals per images (AR^{100} and AR^{1k}).

Implementation details. All architectures in Table 1 are trained end-to-end. The input image is resized such that its shorter side has 800 pixels. We adopt synchronized SGD training on 8 GPUs. A mini-batch involves 2 images per GPU and 256 anchors per image. We use a weight decay of 0.0001 and a momentum of 0.9. The learning rate is 0.02 for the first 30k mini-batches and 0.002 for the next 10k. For all RPN experiments (including baselines), we include the anchor boxes that are outside the image for training, which is unlike [29] where these anchor boxes are ignored. Other implementation details are as in [29]. Training RPN with FPN on 8 GPUs takes about 8 hours on COCO.

[2] https://github.com/kaiminghe/deep-residual-networks
[3] https://github.com/rbgirshick/py-faster-rcnn
[4] https://github.com/caffe2/caffe2

5.1.1 Ablation Experiments

Comparisons with baselines. For fair comparisons with original RPNs [29], we run two baselines (Table 1(a, b)) using the single-scale map of C_4 (the same as [16]) or C_5, both using the same hyper-parameters as ours, including using 5 scale anchors of $\{32^2, 64^2, 128^2, 256^2, 512^2\}$. Table 1 (b) shows no advantage over (a), indicating that a single higher-level feature map is not enough because there is a trade-off between coarser resolutions and stronger semantics.

Placing FPN in RPN improves AR^{1k} to 56.3 (Table 1 (c)), which is **8.0** points increase over the single-scale RPN baseline (Table 1 (a)). In addition, the performance on small objects (AR^{1k}_s) is boosted by a large margin of 12.9 points. Our pyramid representation greatly improves RPN's robustness to object scale variation.

How important is top-down enrichment? Table 1(d) shows the results of our feature pyramid without the top-down pathway. With this modification, the 1×1 lateral connections followed by 3×3 convolutions are attached to the bottom-up pyramid. This architecture simulates the effect of reusing the pyramidal feature hierarchy (Fig. 1(b)).

The results in Table 1(d) are just on par with the RPN baseline and lag far behind ours. We conjecture that this is because there are large semantic gaps between different levels on the bottom-up pyramid (Fig. 1(b)), especially for very deep ResNets. We have also evaluated a variant of Table 1(d) without sharing the parameters of the heads, but observed similarly degraded performance. This issue cannot be simply remedied by level-specific heads.

How important are lateral connections? Table 1(e) shows the ablation results of a top-down feature pyramid without the 1×1 lateral connections. This top-down pyramid has strong semantic features and fine resolutions. But we argue that the locations of these features are not precise, because these maps have been downsampled and upsampled several times. More precise locations of features can be directly passed from the finer levels of the bottom-up maps via the lateral connections to the top-down maps. As a results, FPN has an AR^{1k} score 10 points higher than Table 1(e).

How important are pyramid representations? Instead of resorting to pyramid representations, one can attach the head to the highest-resolution, strongly semantic feature maps of P_2 (*i.e.*, the finest level in our pyramids). Similar to the single-scale baselines, we assign all anchors to the P_2 feature map. This variant (Table 1(f)) is better than the baseline but inferior to our approach. RPN is a sliding window detector with a fixed window size, so scanning over pyramid levels can increase its robustness to scale variance.

In addition, we note that using P_2 alone leads to more anchors (750k, Table 1(f)) caused by its large spatial resolution. This result suggests that a larger number of anchors is not sufficient in itself to improve accuracy.

RPN	feature	# anchors	lateral?	top-down?	AR^{100}	AR^{1k}	AR^{1k}_s	AR^{1k}_m	AR^{1k}_l
(a) baseline on conv4	C_4	47k			36.1	48.3	32.0	58.7	62.2
(b) baseline on conv5	C_5	12k			36.3	44.9	25.3	55.5	64.2
(c) **FPN**	$\{P_k\}$	200k	✓	✓	**44.0**	**56.3**	**44.9**	**63.4**	66.2
Ablation experiments follow:									
(d) bottom-up pyramid	$\{P_k\}$	200k	✓		37.4	49.5	30.5	59.9	**68.0**
(e) top-down pyramid, w/o lateral	$\{P_k\}$	200k		✓	34.5	46.1	26.5	57.4	64.7
(f) only finest level	P_2	750k	✓	✓	38.4	51.3	35.1	59.7	67.6

Table 1. Bounding box proposal results using RPN [29], evaluated on the COCO `minival` set. All models are trained on `trainval35k`. The columns "lateral" and "top-down" denote the presence of lateral and top-down connections, respectively. The column "feature" denotes the feature maps on which the heads are attached. All results are based on ResNet-50 and share the same hyper-parameters.

Fast R-CNN	proposals	feature	head	lateral?	top-down?	AP@0.5	AP	AP_s	AP_m	AP_l
(a) baseline on conv4	RPN, $\{P_k\}$	C_4	conv5			54.7	31.9	15.7	36.5	45.5
(b) baseline on conv5	RPN, $\{P_k\}$	C_5	*2fc*			52.9	28.8	11.9	32.4	43.4
(c) **FPN**	RPN, $\{P_k\}$	$\{P_k\}$	*2fc*	✓	✓	**56.9**	**33.9**	**17.8**	**37.7**	**45.8**
Ablation experiments follow:										
(d) bottom-up pyramid	RPN, $\{P_k\}$	$\{P_k\}$	*2fc*	✓		44.9	24.9	10.9	24.4	38.5
(e) top-down pyramid, w/o lateral	RPN, $\{P_k\}$	$\{P_k\}$	*2fc*		✓	54.0	31.3	13.3	35.2	45.3
(f) only finest level	RPN, $\{P_k\}$	P_2	*2fc*	✓	✓	56.3	33.4	17.3	37.3	45.6

Table 2. Object detection results using **Fast R-CNN** [11] on *a fixed set of proposals* (RPN, $\{P_k\}$, Table 1(c)), evaluated on the COCO `minival` set. Models are trained on the `trainval35k` set. All results are based on ResNet-50 and share the same hyper-parameters.

Faster R-CNN	proposals	feature	head	lateral?	top-down?	AP@0.5	AP	AP_s	AP_m	AP_l
(*) baseline from He *et al.* [16]†	RPN, C_4	C_4	conv5			47.3	26.3	-	-	-
(a) baseline on conv4	RPN, C_4	C_4	conv5			53.1	31.6	13.2	35.6	**47.1**
(b) baseline on conv5	RPN, C_5	C_5	*2fc*			51.7	28.0	9.6	31.9	43.1
(c) **FPN**	RPN, $\{P_k\}$	$\{P_k\}$	*2fc*	✓	✓	**56.9**	**33.9**	**17.8**	**37.7**	45.8

Table 3. Object detection results using **Faster R-CNN** [29] evaluated on the COCO `minival` set. *The backbone network for RPN are consistent with Fast R-CNN.* Models are trained on the `trainval35k` set and use ResNet-50. †Provided by authors of [16].

5.2. Object Detection with Fast/Faster R-CNN

Next we investigate FPN for region-based (non-sliding window) detectors. We evaluate object detection by the COCO-style Average Precision (AP) and PASCAL-style AP (at a single IoU threshold of 0.5). We also report COCO AP on objects of small, medium, and large sizes (namely, AP_s, AP_m, and AP_l) following the definitions in [21].

Implementation details. The input image is resized such that its shorter side has 800 pixels. Synchronized SGD is used to train the model on 8 GPUs. Each mini-batch involves 2 image per GPU and 512 RoIs per image. We use a weight decay of 0.0001 and a momentum of 0.9. The learning rate is 0.02 for the first 60k mini-batches and 0.002 for the next 20k. We use 2000 RoIs per image for training and 1000 for testing. Training Fast R-CNN with FPN takes about 10 hours on the COCO dataset.

5.2.1 Fast R-CNN (on fixed proposals)

To better investigate FPN's effects on the region-based detector alone, we conduct ablations of Fast R-CNN on *a fixed set of proposals*. We choose to freeze the proposals as computed by RPN on FPN (Table 1(c)), because it has good performance on small objects that are to be recognized by the detector. For simplicity we do not share features between Fast R-CNN and RPN, except when specified.

As a ResNet-based Fast R-CNN baseline, following [16], we adopt RoI pooling with an output size of 14×14 and attach all conv5 layers as the hidden layers of the head. This gives an AP of 31.9 in Table 2(a). Table 2(b) is a baseline exploiting an MLP head with 2 hidden *fc* layers, similar to the head in our architecture. It gets an AP of 28.8, indicating that the 2-*fc* head does not give us any orthogonal advantage over the baseline in Table 2(a).

Table 2(c) shows the results of our FPN in Fast R-CNN. Comparing with the baseline in Table 2(a), our method improves AP by 2.0 points and small object AP by 2.1 points. Comparing with the baseline that also adopts a 2*fc* head (Table 2(b)), our method improves AP by 5.1 points.[5] These comparisons indicate that our feature pyramid is superior to single-scale features for a *region-based* object detector.

Table 2(d) and (e) show that removing top-down con-

[5]We expect a stronger architecture of the head [30] will improve upon our results, which is beyond the focus of this paper.

method	backbone	competition	image pyramid	test-dev					test-std				
				$AP_{@.5}$	AP	AP_s	AP_m	AP_l	$AP_{@.5}$	AP	AP_s	AP_m	AP_l
ours, Faster R-CNN on **FPN**	ResNet-101	-		**59.1**	**36.2**	**18.2**	**39.0**	48.2	**58.5**	**35.8**	**17.5**	**38.7**	47.8
*Competition-winning **single-model** results follow:*													
G-RMI†	Inception-ResNet	2016		-	34.7	-	-	-	-	-	-	-	-
AttractioNet‡ [10]	VGG16 + Wide ResNet§	2016	✓	53.4	35.7	15.6	38.0	**52.7**	52.9	35.3	14.7	37.6	**51.9**
Faster R-CNN +++ [16]	ResNet-101	2015	✓	55.7	34.9	15.6	38.7	50.9	-	-	-	-	-
Multipath [40] (on minival)	VGG-16	2015		49.6	31.5	-	-	-	-	-	-	-	-
ION‡ [2]	VGG-16	2015		53.4	31.2	12.8	32.9	45.2	52.9	30.7	11.8	32.8	44.8

Table 4. Comparisons of **single-model** results on the COCO detection benchmark. Some results were not available on the test-std set, so we also include the test-dev results (and for Multipath [40] on minival). †: http://image-net.org/challenges/talks/2016/GRMI-COCO-slidedeck.pdf. ‡: http://mscoco.org/dataset/#detections-leaderboard. §: This entry of AttractioNet [10] adopts VGG-16 for proposals and Wide ResNet [39] for object detection, so is not strictly a single-model result.

nections or removing lateral connections leads to inferior results, similar to what we have observed in the above subsection for RPN. It is noteworthy that removing top-down connections (Table 2(d)) significantly degrades the accuracy, suggesting that Fast R-CNN suffers from using the low-level features at the high-resolution maps.

In Table 2(f), we adopt Fast R-CNN on the single finest scale feature map of P_2. Its result (33.4 AP) is marginally worse than that of using all pyramid levels (33.9 AP, Table 2(c)). We argue that this is because RoI pooling is a warping-like operation, which is less sensitive to the region's scales. Despite the good accuracy of this variant, it is based on the RPN proposals of $\{P_k\}$ and has thus already benefited from the pyramid representation.

5.2.2 Faster R-CNN (on consistent proposals)

In the above we used a fixed set of proposals to investigate the detectors. But in a Faster R-CNN system [29], the RPN and Fast R-CNN must use *the same network backbone* in order to make feature sharing possible. Table 3 shows the comparisons between our method and two baselines, all using *consistent* backbone architectures for RPN and Fast R-CNN. Table 3(a) shows our reproduction of the baseline Faster R-CNN system as described in [16]. Under controlled settings, our FPN (Table 3(c)) is better than this strong baseline by **2.3** points AP and **3.8** points AP@0.5.

Note that Table 3(a) and (b) are baselines that are much stronger than the baseline provided by He *et al.* [16] in Table 3(*). We find the following implementations contribute to the gap: (i) We use an image scale of 800 pixels instead of 600 in [11, 16]; (ii) We train with 512 RoIs per image which accelerate convergence, in contrast to 64 RoIs in [11, 16]; (iii) We use 5 scale anchors instead of 4 in [16] (adding 32^2); (iv) At test time we use 1000 proposals per image instead of 300 in [16]. So comparing with He *et al.*'s ResNet-50 Faster R-CNN baseline in Table 3(*), our method improves AP by 7.6 points and AP@0.5 by 9.6 points.

Sharing features. In the above, for simplicity we do not share the features between RPN and Fast R-CNN. In Table 5, we evaluate sharing features following the 4-step training described in [29]. Similar to [29], we find that sharing features improves accuracy by a small margin. Feature sharing also reduces the testing time.

	ResNet-50		ResNet-101	
share features?	$AP_{@0.5}$	AP	$AP_{@0.5}$	AP
no	56.9	33.9	58.0	35.0
yes	**57.2**	**34.3**	**58.2**	**35.2**

Table 5. More object detection results using Faster R-CNN and our FPNs, evaluated on minival. Sharing features increases train time by 1.5× (using 4-step training [29]), but reduces test time.

Running time. With feature sharing, our FPN-based Faster R-CNN system has inference time of 0.165 seconds per image on a single NVIDIA M40 GPU for ResNet-50, and 0.19 seconds for ResNet-101.[6] As a comparison, the single-scale ResNet-50 baseline in Table 3(a) runs at 0.32 seconds. Our method introduces small extra cost by the extra layers in the FPN, but has a lighter weight head. Overall our system is faster than the ResNet-based Faster R-CNN counterpart. We believe the efficiency and simplicity of our method will benefit future research and applications.

5.2.3 Comparing with COCO Competition Winners

We find that our ResNet-101 model in Table 5 is not sufficiently trained with the default learning rate schedule. So we increase the number of mini-batches by 2× at each learning rate when training the Fast R-CNN step. This increases AP on minival to 35.6, without sharing features. This model is the one we submitted to the COCO detection leaderboard, shown in Table 4. We have not evaluated its feature-sharing version due to limited time, which should be slightly better as implied by Table 5.

Table 4 compares our method with the *single-model* results of the COCO competition winners, including the 2016 winner G-RMI and the 2015 winner Faster R-CNN+++. *Without adding bells and whistles*, our single-model entry has surpassed these strong, heavily engineered competitors.

[6] These runtimes are updated from an earlier version of this paper.

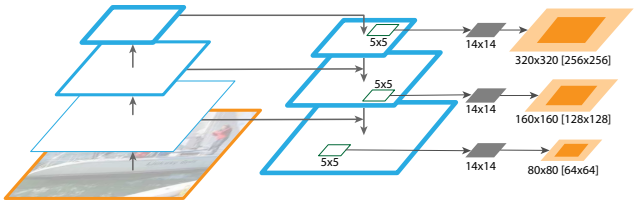


Figure 4. FPN for object segment proposals. The feature pyramid is constructed with identical structure as for object detection. We apply a small MLP on 5×5 windows to generate dense object segments with output dimension of 14×14. Shown in orange are the size of the image regions the mask corresponds to for each pyramid level (levels P_{3-5} are shown here). Both the corresponding image region size (light orange) and canonical object size (dark orange) are shown. Half octaves are handled by an MLP on 7x7 windows ($7 \approx 5\sqrt{2}$), not shown here. Details are in the appendix.

On the `test-dev` set, our method increases over the existing best results by **0.5** points of AP (36.2 *vs*. 35.7) and **3.4** points of AP@0.5 (59.1 *vs*. 55.7). It is worth noting that our method does not rely on image pyramids and only uses a single input image scale, but still has outstanding AP on small-scale objects. This could only be achieved by high-resolution image inputs with previous methods.

Moreover, our method does *not* exploit many popular improvements, such as iterative regression [9], hard negative mining [35], context modeling [16], stronger data augmentation [22], *etc*. These improvements are complementary to FPNs and should boost accuracy further.

Recently, FPN has enabled new top results in *all* tracks of the COCO competition, including detection, instance segmentation, and keypoint estimation. See [14] for details.

6. Extensions: Segmentation Proposals

Our method is a generic pyramid representation and can be used in applications other than object detection. In this section we use FPNs to generate segmentation proposals, following the DeepMask/SharpMask framework [27, 28].

DeepMask/SharpMask were trained on image crops for predicting instance segments and object/non-object scores. At inference time, these models are run convolutionally to generate dense proposals in an image. To generate segments at multiple scales, image pyramids are necessary [27, 28].

It is easy to adapt FPN to generate mask proposals. We use a fully convolutional setup for both training and inference. We construct our feature pyramid as in Sec. 5.1 and set $d = 128$. On top of each level of the feature pyramid, we apply a small 5×5 MLP to predict 14×14 masks and object scores in a fully convolutional fashion, see Fig. 4. Additionally, motivated by the use of 2 scales per octave in the image pyramid of [27, 28], we use a second MLP of input size 7×7 to handle half octaves. The two MLPs play a similar role as anchors in RPN. The architecture is trained end-to-end; full implementation details are given in the appendix.

	image pyramid	AR	AR_s	AR_m	AR_l	time (s)
DeepMask [27]	✓	37.1	15.8	50.1	54.9	0.49
SharpMask [28]	✓	39.8	17.4	53.1	59.1	0.77
InstanceFCN [4]	✓	39.2	–	–	–	1.50†
FPN Mask Results:						
single MLP [5×5]		43.4	32.5	49.2	53.7	**0.15**
single MLP [7×7]		43.5	30.0	49.6	57.8	0.19
dual MLP [5×5, 7×7]		45.7	31.9	51.5	60.8	0.24
+ 2x mask resolution		46.7	31.7	53.1	63.2	0.25
+ 2x train schedule		**48.1**	**32.6**	**54.2**	**65.6**	0.25

Table 6. Instance segmentation proposals evaluated on the first 5k COCO `val` images. All models are trained on the `train` set. DeepMask, SharpMask, and FPN use ResNet-50 while InstanceFCN uses VGG-16. DeepMask and SharpMask performance is computed with models available from https://github.com/facebookresearch/deepmask (both are the 'zoom' variants). †Runtimes are measured on an NVIDIA M40 GPU, except the InstanceFCN timing which is based on the slower K40.

6.1. Segmentation Proposal Results

Results are shown in Table 6. We report segment AR and segment AR on small, medium, and large objects, always for 1000 proposals. Our baseline FPN model with a single 5×5 MLP achieves an AR of 43.4. Switching to a slightly larger 7×7 MLP leaves accuracy largely unchanged. Using both MLPs together increases accuracy to 45.7 AR. Increasing mask output size from 14×14 to 28×28 increases AR another point (larger sizes begin to degrade accuracy). Finally, doubling the training iterations increases AR to 48.1.

We also report comparisons to DeepMask [27], SharpMask [28], and InstanceFCN [4], the previous state of the art methods in mask proposal generation. We outperform the accuracy of these approaches by over **8.3** points AR. In particular, we nearly double the accuracy on small objects.

Existing mask proposal methods [27, 28, 4] are based on densely sampled image pyramids (*e.g*., scaled by $2^{\{-2:0.5:1\}}$ in [27, 28]), making them computationally expensive. Our approach, based on FPNs, is substantially faster (our models run at 4 to 6 fps). These results demonstrate that our model is a generic feature extractor and can replace image pyramids for other multi-scale detection problems.

7. Conclusion

We have presented a clean and simple framework for building feature pyramids inside ConvNets. Our method shows significant improvements over several strong baselines and competition winners. Thus, it provides a practical solution for research and applications of feature pyramids, without the need of computing image pyramids. Finally, our study suggests that despite the strong representational power of deep ConvNets and their implicit robustness to scale variation, it is still critical to explicitly address multi-scale problems using pyramid representations.

References

[1] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden. Pyramid methods in image processing. *RCA engineer*, 1984.

[2] S. Bell, C. L. Zitnick, K. Bala, and R. Girshick. Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks. In *CVPR*, 2016.

[3] Z. Cai, Q. Fan, R. S. Feris, and N. Vasconcelos. A unified multi-scale deep convolutional neural network for fast object detection. In *ECCV*, 2016.

[4] J. Dai, K. He, Y. Li, S. Ren, and J. Sun. Instance-sensitive fully convolutional networks. In *ECCV*, 2016.

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

[6] P. Dollár, R. Appel, S. Belongie, and P. Perona. Fast feature pyramids for object detection. *TPAMI*, 2014.

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

[8] G. Ghiasi and C. C. Fowlkes. Laplacian pyramid reconstruction and refinement for semantic segmentation. In *ECCV*, 2016.

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

[10] S. Gidaris and N. Komodakis. Attend refine repeat: Active box proposal generation via in-out localization. In *BMVC*, 2016.

[11] R. Girshick. Fast R-CNN. In *ICCV*, 2015.

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

[13] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik. Hypercolumns for object segmentation and fine-grained localization. In *CVPR*, 2015.

[14] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. *arXiv:1703.06870*, 2017.

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

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

[17] S. Honari, J. Yosinski, P. Vincent, and C. Pal. Recombinator networks: Learning coarse-to-fine feature aggregation. In *CVPR*, 2016.

[18] T. Kong, A. Yao, Y. Chen, and F. Sun. Hypernet: Towards accurate region proposal generation and joint object detection. In *CVPR*, 2016.

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

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

[21] 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.

[22] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, and S. Reed. SSD: Single shot multibox detector. In *ECCV*, 2016.

[23] W. Liu, A. Rabinovich, and A. C. Berg. ParseNet: Looking wider to see better. In *ICLR workshop*, 2016.

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

[25] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 2004.

[26] A. Newell, K. Yang, and J. Deng. Stacked hourglass networks for human pose estimation. In *ECCV*, 2016.

[27] P. O. Pinheiro, R. Collobert, and P. Dollar. Learning to segment object candidates. In *NIPS*, 2015.

[28] P. O. Pinheiro, T.-Y. Lin, R. Collobert, and P. Dollár. Learning to refine object segments. In *ECCV*, 2016.

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

[30] S. Ren, K. He, R. Girshick, X. Zhang, and J. Sun. Object detection networks on convolutional feature maps. *PAMI*, 2016.

[31] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015.

[32] H. Rowley, S. Baluja, and T. Kanade. Human face detection in visual scenes. Technical Report CMU-CS-95-158R, Carnegie Mellon University, 1995.

[33] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 2015.

[34] 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.

[35] A. Shrivastava, A. Gupta, and R. Girshick. Training region-based object detectors with online hard example mining. In *CVPR*, 2016.

[36] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

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

[38] 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.

[39] S. Zagoruyko and N. Komodakis. Wide residual networks. In *BMVC*, 2016.

[40] S. Zagoruyko, A. Lerer, T.-Y. Lin, P. O. Pinheiro, S. Gross, S. Chintala, and P. Dollár. A multipath network for object detection. In *BMVC*, 2016.