Feature Pyramid Grids

Kai Chen[1,2*] Yuhang Cao[1] Chen Change Loy[1] Dahua Lin[1] Christoph Feichtenhofer[2]

[1]The Chinese University of Hong Kong [2]Facebook AI Research (FAIR)

Abstract

Feature pyramid networks (FPN) have been widely adopted in the object detection literature to improve feature representations for better handling of variations in scale. In this paper, we present Feature Pyramid Grids (FPG), a simple extension to FPN, that represents the feature scale-space as a regular grid of parallel bottom-up pathways which are fused by multi-directional lateral connections between them. FPG is simple and flexible, which only adds a small overhead to regular, single pathway FPN while significantly increasing its performance. In addition to its general and simple structure over complicated structures that have been found with neural architecture search, it also compares favorably against such approaches, providing higher accuracy and speed. We hope that FPG with its simple and effective nature can serve as a strong baseline for future work in object recognition.

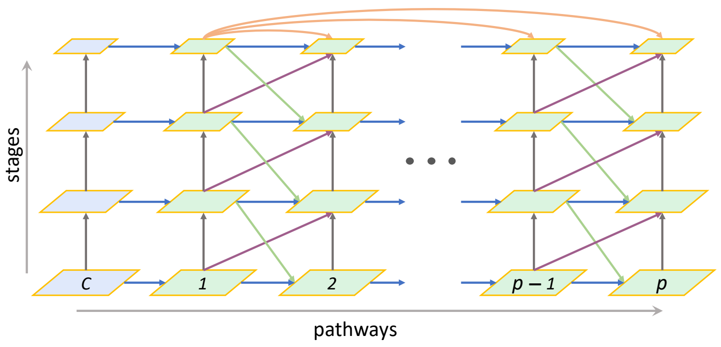


Figure 1: **A Feature Pyramid Grid (FPG)** connects the ***backbone features***, C, of a ConvNet with a regular structure of p parallel top-down pyramid ***pathways*** which are fused by multi-directional *lateral connections*, AcrossSame →, AcrossUp ↗, AcrossDown ↘, and AcrossSkip ↷.

1. Introduction

It seems trivial how human perception can simultaneously recognize visual information across various levels of different resolution. For machine perception, recognizing objects at various scales has been a classical challenge in visual recognition over decades [2, 1, 17, 29, 31]. Numerous methods have been developed to build pyramid representations [2, 1, 17] as an effective way to model the scale-space, by building a hierarchical pyramid ranging from large to small image scales. Such classical pyramid representations are typically built by subsequent filtering (blurring) and subsampling operations applied to the image.

In recent deep learning approaches, a bottom-up pathway is inherently built by ConvNets [20] that hierarchically abstract information from higher to lower resolution in deeper layers, also by hierarchical filtering and subsampling. For object detection tasks, Feature Pyramid Networks (FPN) [21], a simple yet effective representation for multi-scale features has become popular. FPN augments ConvNets with a second top-down pathway and lateral connections to enrich high-resolution features with semantic information from deeper, but lower-resolution features.

In this paper, we present Feature Pyramid Grids (FPG), a simple multi-pathway extension to FPN [21], that represents the feature scale-space as a regular grid of parallel pathways fused by multi-directional lateral connections between them, as shown in Fig. 1. FPG enriches the hierarchical feature representation built internally in the backbone pathway of a ConvNet with *multiple* pyramid pathways in parallel. Therefore, in concept, FPG is a simple extension of FPN from one to p pathways.

Differently from FPN, the individual pathways are built in a bottom-up manner, similar to the backbone pathway that goes from the input image a prediction output. To form a *grid of feature pyramids*, the individual pyramid pathways are intertwined with various lateral connections, both across-scale as well as within-scale to enable information exchange across all levels. We categorize these *lateral connections* into four types, AcrossSame →, AcrossUp ↗, AcrossDown ↘, and AcrossSkip ↷, shown in Fig. 1. Note, for clarity, AcrossSkip connections are only shown on the top of the figure, but are used densely throughout the grid.

In the paper we show that FPG is simple, efficient, and generalizes well across detectors. We hope that it can serve as a strong baseline for future work in object recognition.

*Work done during an internship at Facebook AI Research.

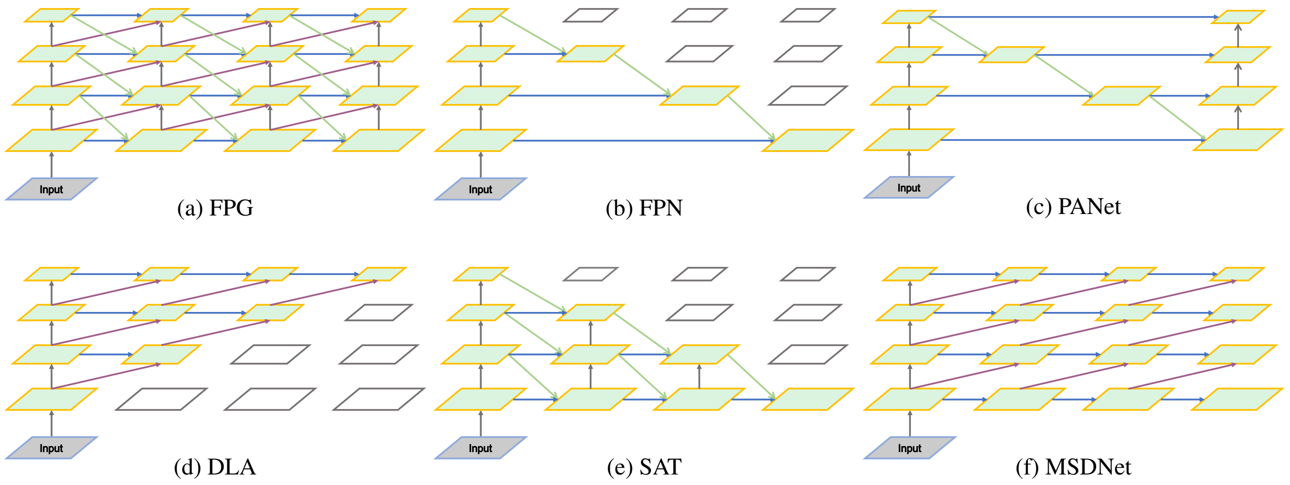


Figure 2: Feature Pyramid Grids (a) in the light of conventional FPN architectures (b-f) represented with the same framework. We show feature pyramids of 4 different scales and 4 **pathways**, omitting backbone features for simplicity. At an abstract level, FPG can be seen as a dense form of these approaches with some of its connections set to ”none” or ”identity”. Skip connections in (a) and (f) are omitted.

There exist recent efforts [7, 41] of applying Neural Architecture Search (NAS) to find better feature pyramid representations, that are particularly related to our approach. As a closely related work, NAS-FPN [7] defines a search space for the modular pyramidal architecture and adopts reinforcement learning to search the best performing one. More recently, Auto-FPN [41] has proposed other search spaces for both FPN and the box head.

These search-based approaches have indeed shown new levels of performance that outperform conventional, manually designed FPN structures. However, in the search of new architectures, there are several drawbacks unique to NAS:

(i) The cost incurred by NAS is large, implicating up to thousands of TPU hours [7] to find an optimal architecture.

(ii) The network structure is often complicated hence not very comprehensible, limiting the adoption of the models.

(iii) The discovered architecture may not generalize well to other detection frameworks. To give an example, NAS-FPN achieves a good performance on RetinaNet (which it was searched for), but is suboptimal for other detectors, as will be shown in our experiments.

In our experimental evaluation, we apply FPG to single-stage (RetinaNet) [22], two-stage (Faster R-CNN, Mask R-CNN) [33, 9] and cascaded (Cascade R-CNN) [4] detectors. Under similar computational cost, FPG performs favorably to FPN or NAS-FPN on various models and settings. Concretely, adopting the same setting as in NAS-FPN, FPG achieves 0.2%, 1.5%, 2.3% and 2.2% higher mean Average Precision (mAP) than NAS-FPN on RetinaNet, Faster R-CNN, Mask R-CNN and Cascade R-CNN, respectively. To demonstrate further generalizability, we apply FPG to the panoptic segmentation task [16] where simply replacing the FPN of Panoptic FPN [15] with FPG improves PQ by 1.3% and 0.5% over FPN and NAS-FPN, respectively.

Conceptually, the idea of FPG is generic, and it can be instantiated with various pathway and lateral connection designs as well as implementation specifics. On an abstract architecture level, many conventional FPN variants can be represented by the FPG idea.

In Fig. 2 we draw comparisons to several popular feature pyramid designs from the literature. From Fig. 2b to 2f, we show Feature Pyramid Network (FPN) [21], Path Aggregation Network (PANet) [26], Deep Layer Aggregation (DLA) [42], Scale Aggregation Topology (SAT) [8] and Multi-Scale DenseNet (MSDNet) [13]. They are all related, and our goal with FPG is to create a complete space that covers all of these instantiations, *on a high level*. Our experiments systematically analyze the importance of key components (parallel pathways and lateral connections) in feature pyramid design using a unified FPG framework, with the aim of striking a good trade-off between accuracy and computation.

2. Related Work

Handcrafted FPN architectures. Scale variation is a well-known challenge for instance-level recognition tasks, and building pyramidal representations have been an effective way to process visual information across various image resolutions, in classical computer vision applications [2, 1, 17, 31], and also in deep learning based approaches [27, 3, 21, 18, 13, 26, 42, 43].

SSD [27] and MS-CNN [3] utilize multi-level feature maps to make predictions, but no aggregation is performed across different feature levels. FPN [21] is the current leading paradigm for learning feature representation of different levels through the top-down pathway and lateral connections. Similarly, RON [18] introduces reverse connections to pass information from high-level to low-level features. Although MSDNet [13] is not designed for FPN, it maintains coarse and fine level features throughout the network with a two-dimensional multi-scale network architecture. HRNet [35] also maintains high-resolution representations through the whole backbone feedforward process. PANet [26] extends FPN by introducing an extra bottom-up pathway to boost information flow. DLA [42] further deepens the representation by nonlinear and progressive fusion. M2Det [43] employs multiple U-Net to pursue a more suitable feature representation for object detection.

In relation to these previous efforts, FPG tries to formalize a unified grid structure that could potentially generalize many of these FPN variations within a systematic multi-pathway structure.

NAS-based FPN architectures. NAS automatically searches for efficient and effective architectures on a specific task. It shows promising results on image classification [32, 25, 37], and is also applied to other downstream tasks [5, 30, 28, 7, 41, 24]. Some methods aim at discovering better FPN architectures. NAS-FPN [7] searches the construction of merging cells, *i.e.*, how to merge features at different scales. It achieves significant accuracyf improvements with a highly complicated wiring pattern (*i.e.* architecture). Auto-FPN [41] searches the architecture of both the FPN and head. It defines a fully connected search space with various dilated convolution operations, resulting in a more lightweight solution than NAS-FPN. Unlike NAS-FPN or FPG, the pathways in Auto-FPN is fixed and the connections of different stacks are not the same, thus making it not easily scalable.

In contrast to NAS-based FPN architectures, FPG can be seen as a more unified approach to feature pyramid representations, which is simple, intuitive and easy to extend.

3. Feature Pyramid Grids

Our objective in this paper is to design a simple and general multi-pathway feature pyramid representation. Building upon FPN [21], we aim to use the hierarchical feature representation built internally by ConvNets and enrich it with multiple pathways and lateral connections between them, to form a regular *Feature Pyramid Grid* (FPG). The concept is illustrated in Fig. 1.

Our generic grid has a backbone pathway (Sec. 3.1) and multiple pyramid pathways (Sec. 3.2), which are fused by lateral connections (Sec. 3.3) to define an FPG network.

3.1. Backbone pathway

The backbone pathway can be the hierarchical feature representation of any ConvNet for image classification (*e.g.*, [19, 34, 36, 10]). This pathway is identical to what is used as the bottom-up pathway in FPN [21]. It has feature maps of progressively smaller scales from the input image to the output. As in [10, 21], feature tensors with the same scale belong to a network stage and the last feature map of each stage is denoted as C_i, where i corresponds to the stage within the backbone hierarchy. The spatial stride of feature tensors w.r.t. the input increases from early to deeper stages, as is common in image classification [19, 34, 36, 10].

3.2. Pyramid pathways

The deeper backbone stages, closer to the classification layer of the network represent high-level semantics, but at low resolution, while the features in early stages are only weakly related to semantics, but, on the other hand, have high localization accuracy due to their fine resolution. The objective of the pyramid pathways is to build fine resolution features with strong semantic information. A single pyramid pathway is defined by the *top-down pathway* in the FPN [21] structure. This pathway consecutively upsamples deeper features of lower resolution to higher resolution of early stages, aiming to propagate semantic information backwards towards the network input, in parallel to the backbone (*i.e.*, feedforward) pathway.

Multiple pyramid pathways. FPG extends this idea by having *multiple*, $p > 1$, pyramid pathways in parallel. Our intention here is to enrich the capacity of the network to build a powerful representation with fine resolution across spatial dimensions and high discriminative ability, by employing multiple pyramid pathways in parallel. A typical value is $p = 9$ parallel pathways in our experiments. Slightly different from FPN, however we build the pyramid pathways in a *bottom-up* manner, in parallel to the backbone pathway (and the first highest resolution pyramid feature is taken from the corresponding backbone stage). Connections in pyramid pathways are denoted as *SameUp*. The presence of multiple pathways is key to the FPG concept (Fig. 1) since it allows the network to build stronger pyramid features as will be demonstrated in our experiments. To form a Feature Pyramid Grid, the p individual pyramid pathways are intertwined with various lateral connections.

Low channel capacity. Following the efficient design of FPN [21], we aim to make the pyramid pathways lightweight by reducing their channel capacity. Concretely, the pyramid pathways use a significantly lower channel capacity than the number of channels of the final stage in the backbone pathway. The typical value is 256 in FPN. Notice that the computation cost (floating-number operations, or

FLOPs) of a weight layer scales *quadraticly* with its channel dimensions (*i.e.* width). Therefore, reducing the channel capacity in the pyramid pathways can make multiple pathways very computationally-effective as we will demonstrate in our experiments.

3.3. Lateral connections

The aim of lateral connections is to enrich features with multi-directional (semantic) information flow in the scale space, and allow complex hierarchical feature learning across different scales.

We are using across-scale as well as within-scale connections between adjacent pathways. Again, this extends FPN that uses lateral connections from the backbone pathway into the feature pyramid built by the top-down pathway. In relation to this, our p parallel pyramid pathways with the lateral connections between define a Feature Pyramid Grid.

We categorize our lateral connections into 4 different categories according to their starting and ending feature stages, which are denoted as:

- Across-pathway same-stage (*AcrossSame*, $\rightarrow$)
- Across-pathway bottom-up connection (*AcrossUp*, $\nearrow$)
- Across-pathway top-down (*AcrossDown*, $\searrow$)
- Across-pathway skip connection (*Skip*, $\curvearrowright$).

We describe how these connections are implemented within the context of a concrete instantiation of FPG next.

3.4. Instantiations

Our idea of FPG is generic, and it can be instantiated with different pathway and lateral connection designs as well as implementation specifics. Here, we describe our instantiations of the network architectures.

Backbone pathway. The backbone 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 (*i.e.* the spatial stride between stages). Taking ResNet [10] for example, we adopt the same scheme as in FPN and use the output feature map of each stage's last residual block to represent the pyramid levels, denoted as $\{C_2, C_3, C_4, C_5\}$.

Pyramid pathways. Similar to the backbone pathway, pyramid pathways represent information across scales. We follow a simple design for building these in a bottom-up manner, starting from the highest resolution stage to the lowest. The first feature map of the pathway is formed by a 1×1 lateral connection from the corresponding high-resolution backbone or pyramid stage. Then we use subsampling to create each lower-level feature map in the pyramid pathway by using a 3×3 convolution width stride 2, Therefore, in each pathway, the feature hierarchy consists of multi-scale feature maps with the same scaling stride for different stages as in the backbone pathway.

Lateral connections. Our lateral connections fuse between the pathways into multiple directions. We employ across-pathway lateral, bottom-up, and top-down connections between adjacent pathways, and skip connections between the first pathway and all other pathways. Identically as for building the p parallel pathways, we employ element-wise *Sum* as the fusion function for all lateral connections [21].

- *AcrossSame*, $\rightarrow$

  Similar to FPN and PANet, we introduce lateral connections to connect the same-level features across pathways. We attach a 1×1 lateral convolution on each feature map to project the features and fuse them with the corresponding feature map in the adjacent pathway.

- *AcrossUp*, $\nearrow$

  In order to shorten the path from low-level features in shallow pathways to high-level features in deep pathways, we introduce direct connections to build the across-level bottom-up pathway. The low-level feature map is downsampled to half size by a 3×3 stride-2 convolution and then fused with the higher-level one.

- *AcrossDown*, $\searrow$

  Similar to our bottom-up information stream within each pathway, we aim for a top-down flow of information by incorporating *AcrossDown* connections. Firstly we upsample the high-level feature maps by a scaling factor of 2 with nearest interpolation, and then use a 3×3 convolution to make $AcrossDown$ learnable. The upsampled features are merged with the low-level features using summation.

- *Skip*, $\curvearrowright$

  To ease the training of such a wide feature pyramid grid, we add skip connections, *e.g.*, 1×1 convolution, between same level of the first pathway and each later pathway.

3.5. Grid details

Given a feature hierarchy in the backbone pathway, *e.g.*, $\{C_2, C_3, C_4, C_5\}$ with strides of $\{4, 8, 16, 32\}$ respectively, as in FPN, we first use 1×1 convolutions to uniformly reduce the channel capacity by β times the width of the highest feature map in the backbone pathway (*e.g.*, 256 for a ResNet with a maximum of width of 2048 and $\beta = 1/8$), producing $\{P_2^1, P_3^1, P_4^1, P_5^1\}$. Similar to FCOS [38], we produce P_6^1 from P_5^1. Then we apply the same topology to each following pathway, until the last one $\{P_2^p, P_3^p, P_4^p, P_5^p, P_6^p\}$.

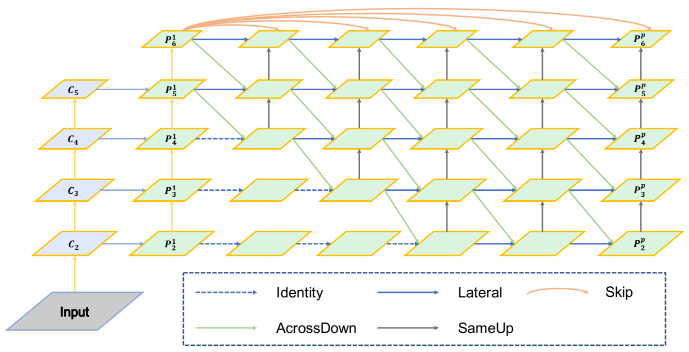


Figure 3: The final contracted architecture of FPG, in which *AcrossUp* is not used and the lower triangular is truncated. We only show the skip connections of the highest level for simplicity.

We follow the standard approach of using $P_2 \sim P_6$ for Faster R-CNN and Mask R-CNN, and $P_3 \sim P_7$ for RetinaNet [22], everything else is identical across detectors.

Following [7], each convolution block in above lateral connections consists of a ReLU [19], a convolution layer, and a BatchNorm [14] layer. Those connections are not shared across different pathways. After the last pathway, we append a vanilla 3×3 convolution layer after each merged feature map to output the final feature map.

We aim for the simplest possible design of FPG. We think that adopting advanced upsampling and downsampling operators such as [39, 6], separable-convolution [11, 41], designing more advanced blocks or fusion strategies (*e.g.* using attention [12, 7]), may further boost the system-level performance, but is not the focus of this work.

Grid contraction. Our ablation studies suggest that the regular design of FPG can be simplified for better computation/accuracy trade-off. This will be demonstrated in our experiments, but for now, we show a more efficient version of FPG that reduces some stages without sacrificing significant accuracy. First, there are two bottom-up streams in our design: *SameUp* and *AcrossUp*. Our ablation analysis in Sec. 4.5 reveals that removing *AcrossUp* has no significant impact, possibly because *SameUp* is sufficient to provide the information flow from low-level to high-level ones which is expected to be less rich in semantic information, and therefore might require lower representation capacity. Second, we also found that reducing connections for high resolution feature maps can be without sacrificing performance. The straight-forward intuition is that low-level feature maps do not need deep transforms, while having a large memory footprint. Specifically, we truncate the lower triangular part of the grid to conserve computation. The final contracted FPG architecture is illustrated in Fig. 3.

4. Experiments

We perform experiments on various object recognition tasks, including object detection, instance segmentation and panoptic segmentation.

4.1. Experimental Setup

Dataset and evaluation metric. We conduct experiments on the MSCOCO 2017 dataset [23]. For all tasks, models are trained on the `train` split and results are reported on `val` splits. We also show our main results on `test-dev`. Evaluation follows standard COCO mAP metrics.

Implementation details. For object detection and instance segmentation, we experiment with two different augmentation settings, denoted as `crop-aug` and `std-aug`. `crop-aug` is the setting introduced in NAS-FPN [7], which randomly resizes and crops input images to a fixed size of 640×640. We utilize 8 GPUs for training and use a batch-size of 8 images on each GPU, resulting in a total batch size of 64. The training lasts for 50 epochs. The initial learning rate is set to 0.08, and then decays by 0.1 after 30 and 40 epochs. BN layers are not frozen in the Pyramid but frozen in other components.

`std-aug` is the standard augmentation procedure in the original publications of the detectors used [33, 9, 22, 4]. It resizes input images to a maximum size of 1333×800 without changing the aspect ratio. Therefore it requires more GPU memory for training. We use 16 GPUs for training and the mini-batch size is 1 image per GPU (so the total mini-batch size is 16). Models are trained for 12 epochs with an initial learning rate of 0.02 and decreased by 0.1 after 8 and 11 epochs. BN statistics are synchronized across GPUs for the Pyramid, and frozen in the ImageNet pre-trained backbone, pathways. We adopt ResNet-50 for the `crop-aug` setting experiments and ResNet-50/101 for the `std-aug` setting. We set the weight decay to 0.0001 and momentum to 0.9. For panoptic segmentation, we simply adopt the default setting in Detectron2 [40], without any modification.

For inference we exactly follow the standard settings in the original architectures [33, 9, 22, 4, 7, 15].

We denote the combination of number of pathways p and the common pathway channel width w as $p@w$. For example, using this terminology 9@256 indicates 9 pathways of channel width 256 for all pyramid layers.

Efficiency metrics. We report single image floating point operations (FLOPs) as a basic *unit* of measuring computational cost, as well as inference-time computational cost which roughly proportional to this, up to implementation and hardware specifics. The inference time is measured by frames-per-second (fps) on a single Tesla V100, including the feedforward and all the pre- and post-processing time.

Detector	Pyramid	FLOPs (G)	Params (M)	fps	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
RetinaNet R-50	FPN 1 @ 256	95.68	37.76	18.1	0.370	0.559	0.397	0.161	0.411	0.515
	NAS-FPN 7 @ 256	138.75	59.74	18.9	0.398	0.585	0.426	0.176	0.448	0.544
	FPG 9 @ 128	95.94	40.10	17.4	0.390	0.582	0.419	0.173	0.439	0.533
	FPG 9 @ 256	136.00	72.50	16.7	**0.400**	0.591	0.429	0.182	0.450	0.547
Faster R-CNN R-50	FPN 1 @ 256	91.42	41.53	28.1	0.376	0.584	0.407	0.184	0.407	0.508
	NAS-FPN 7 @ 256	265.29	68.17	19.1	0.399	0.588	0.433	0.188	0.438	0.544
	FPG 9 @ 128	99.11	42.14	20.8	0.400	0.599	0.435	0.200	0.436	0.538
	FPG 9 @ 256	254.11	79.80	16.5	**0.414**	0.614	0.451	0.215	0.448	0.548
Mask R-CNN R-50	FPN 1 @ 256	293.35	44.18	17.7	0.386	0.592	0.419	0.187	0.414	0.524
	NAS-FPN 7 @ 256	333.80	70.81	15.0	0.401	0.579	0.443	0.190	0.457	0.581
	FPG 9 @ 128	161.84	44.49	15.1	0.409	0.605	0.446	0.209	0.444	0.546
	FPG 9 @ 256	322.62	82.45	12.7	**0.424**	0.621	0.463	0.225	0.458	0.560
Cascade R-CNN R-50	FPN 1 @ 256	119.07	69.17	22.0	0.406	0.585	0.439	0.195	0.434	0.555
	Nas-FPN 7 @ 256	292.93	95.80	15.9	0.416	0.583	0.451	0.191	0.458	0.573
	FPG 9 @ 128	113.90	56.93	17.8	0.425	0.600	0.460	0.214	0.459	0.573
	FPG 9 @ 256	281.75	107.44	13.8	**0.438**	0.615	0.476	0.232	0.472	0.582

Table 1: **Object detection** mAP on COCO `test-dev`. Results of different detectors on COCO with the *crop-aug* setting, FLOPs and inference time (fps) is reported on a single image of size 640x640.

Detector	Pyramid	FLOPs (G)	Params (M)	fps	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Faster R-CNN R-50	FPN 1 @ 256	214.84	41.53	17.3	0.366	0.588	0.396	0.216	0.398	0.450
	NAS-FPN 7 @ 256	666.86	68.17	9.6	0.390	0.595	0.424	0.224	0.426	0.478
	FPG 9 @ 128	245.07	42.14	13.3	0.380	0.594	0.412	0.221	0.407	0.464
	FPG 9 @ 256	637.79	79.80	8.4	**0.392**	0.608	0.427	0.227	0.419	0.484
Faster R-CNN R-101	FPN 1 @ 256	293.96	60.53	14.2	0.388	0.609	0.423	0.223	0.422	0.486
	NAS-FPN 7 @ 256	745.97	87.16	8.3	0.403	0.612	0.438	0.231	0.439	0.501
	FPG 9 @ 128	324.19	61.13	11.8	0.395	0.610	0.430	0.229	0.424	0.492
	FPG 9 @ 256	716.90	98.80	7.6	**0.406**	0.622	0.443	0.234	0.435	0.506
Mask R-CNN R-50	FPN 1 @ 256	283.35	44.18	12.0	0.374	0.593	0.407	0.220	0.406	0.463
	NAS-FPN 7 @ 256	735.37	70.81	8.0	0.396	0.598	0.433	0.228	0.427	0.484
	FPG 9 @ 128	307.80	44.49	10.4	0.390	0.599	0.424	0.228	0.418	0.484
	FPG 9 @ 256	706.29	82.45	7.2	**0.403**	0.612	0.442	0.237	0.428	0.497
Mask R-CNN R-101	FPN 1 @ 256	362.46	63.17	10.7	0.397	0.616	0.432	0.230	0.432	0.497
	NAS-FPN 7 @ 256	814.48	89.80	7.3	0.405	0.608	0.442	0.234	0.437	0.502
	FPG 9 @ 128	386.91	63.48	9.3	0.405	0.615	0.443	0.235	0.436	0.502
	FPG 9 @ 256	785.41	101.44	6.5	**0.416**	0.627	0.455	0.241	0.445	0.516

Table 2: **Object detection** mAP on COCO `test-dev`. Results of different detectors on COCO with the *std-aug* setting., FLOPs and inference time (fps) is reported on a single image of size 1280x832.

4.2. Main Results on Object Detection

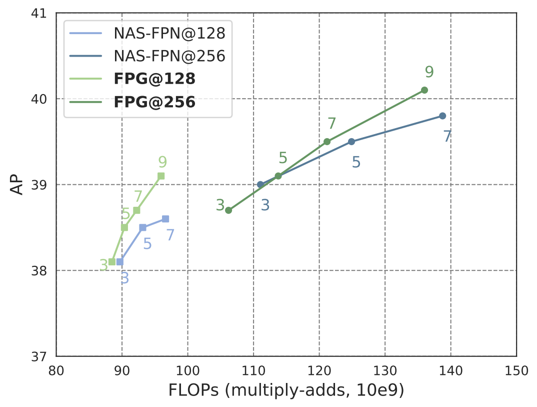
Overall results. We apply FPG to various detectors with the `crop-aug` and `std-aug` settings. We report the performance of 2 different architectures: *FPG (9@256)* and *FPG (9@128)*. FPG (9@256) has comparable FLOPs with NAS-FPN (7@256), and FPG (9@128) is as lightweight version of FPG that roughly matches the computational cost of the default FPN (1@256).

The results of the `crop-aug` setting are shown in Table 1, for four different detection architectures. On all, single-stage (RetinaNet) [22], two-stage (Faster R-CNN, Mask R-CNN) [33, 9] and cascaded (Cascade R-CNN) [4] detectors, FPG outperforms FPN by a solid margin, and also achieves better performance than NAS-FPN.

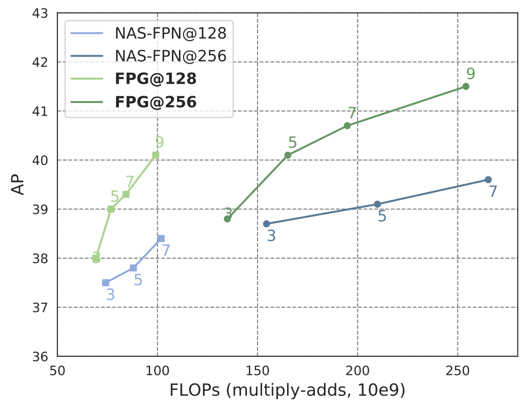
Compared to FPN, our FPG (9@128) improves the box AP by +2.0, +2.4, +2.3, and +1.9 mAP on RetinaNet, Faster R-CNN, Mask R-CNN, and Cascade R-CNN, respectively.

Compared with NAS-FPN (7@256), FPG (9@256) achieves +0.2, +1.5, +2.3 and +2.2 higher mAP on those four detectors, while maintaining slightly less FLOPs. Without bells and whistles, FPG (9@256) obtains an mAP of 41.4% on Faster R-CNN and 43.8% on Cascade R-CNN while using a ResNet-50 backbone.

It is also worth noting that NAS-FPN performs comparably to FPG on RetinaNet, but achieves inferior results on the other detection systems. As we observe higher gains over NAS-FPN in multi-stage detectors, this suggests that the NAS-FPN architecture found for the single-stage detection architecture (*i.e*. RetinaNet) might not generalize well to multi-stage detectors. The simple FPG exhibits *good generalization across all detection systems*. The results of the `std-aug` setting are shown in Table 2 and are consistent with our findings for the `crop-aug` setting.



(a) RetinaNet.



(b) Faster R-CNN

Figure 4: The efficiency-accuracy trade-off for increasing number of pathways (FPG) and stacks (NAS-FPN) from 3 to 9, using different feature dimensions (128/256). We compare NAS-FPN and FPG on two different models: RetinaNet (a) and Faster R-CNN (b). The results are under the `crop-aug` setting and show the performance on COCO *val2017*.

Efficiency *vs*. accuracy trade-off. Feature pyramid architectures make it easy to configure different model capacities. By adjusting the number of pyramid pathways p and the pathway width w we can trade off the efficiency with accuracy and obtain a set of FPG networks, from lightweight to heavy computational cost. We apply the same principle to FPN and NAS-FPN and investigate the computation/accuracy trade-off next.

Figure 4 shows the effects of multiple (*i.e*. 3, 5, 7, 9) FPG pyramid pathways or more NAS-FPN stacks, as well as varying the channel width (128, 256). For both NAS-FPN and FPG, adopting a larger channel width improves the accuracy while resulting in more FLOPs. Under limited channel capacity (@128), the gap between NAS-FPN and FPG is even larger. When the pyramid width is set to 128, the curve of FPG is significant above that of NAS-FPN. When comparing Fig. 4b, we observe that FPG achieves significantly better efficiency-accuracy trade-off for Faster R-CNN detectors for which NAS-FPN was not searched (it was searched on RetinaNet), illustrating the generalization of FPG to different architectures.

Figure 5 shows that adopting more FPG pyramid pathways or more NAS-FPN stacks is beneficial, but for vanilla FPN more than 3 top-down pathways are not. Adding more than 3 pathways on FPN will instead decrease the mAP. Overall, FPG achieves a better trade-off than FPN and NAS-FPN. For example, FPG (9@256) achieves a higher mAP than NAS-FPN (7@256) with fewer FLOPs.

4.3. Main Results on Instance Segmentation

We also report the instance segmentation results of Mask R-CNN in Table 3. The setting is the same as in Table 1. FPG (9@256) achieves +2.7 higher mask AP than FPN and 1.6 higher mask AP than NAS-FPN (7@256), demonstrating generalization of FPG across different tasks.

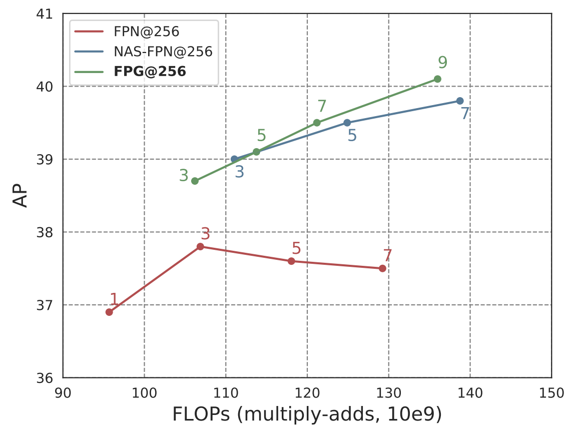


Figure 5: The efficiency-accuracy trade-off by adjusting number of pathways (FPN, FPG) and stacks (NAS-FPN). We compare FPN, NAS-FPN and FPG on RetinaNet.

Neck	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
FPN	0.345	0.552	0.368	0.134	0.372	0.540
NAS-FPN	0.356	0.552	0.385	0.131	0.393	0.569
FPG	**0.372**	**0.584**	**0.398**	**0.159**	**0.403**	**0.570**

Table 3: **Instance segmentation** *mask* AP on COCO `val2017`. FPG provides solid improvements over the FPN and NAS-FPN variants.

Neck	PQ	SQ	RQ	PQ^{Th}	PQ^{St}	mIoU	AP
FPN	0.394	0.778	0.483	0.459	0.296	0.412	0.347
NAS-FPN	0.402	0.783	0.488	0.466	0.307	0.421	0.359
FPG	**0.407**	**0.786**	**0.495**	**0.472**	**0.310**	**0.424**	**0.362**

Table 4: **Panoptic segmentation** on COCO `val2017`. **FPG** performs better than the FPN and NAS-FPN variants.

AcrossDown	AcrossUp	SameUp	AcrossSkip	Contraction	FLOPs	Params	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
✓	✓	✓	✓		173.32	104.51	**0.401**	0.590	0.430	0.199	0.456	0.567
	✓	✓	✓		128.14	83.26	0.355	0.529	0.382	0.140	0.397	0.537
✓		✓	✓		162.01	83.26	0.401	0.590	0.429	0.196	0.461	0.564
✓	✓		✓		162.01	83.26	0.396	0.586	0.423	0.187	0.454	0.559
✓	✓	✓			162.21	101.53	0.395	0.584	0.422	0.186	0.453	0.566
✓		✓	✓	✓	136.00	72.50	**0.401**	0.592	0.427	0.194	0.457	0.571

Table 5: **Ablations**. We study the effectiveness of each component of FPG on `val2019` and report *box* AP.

4.4. Main Results on Panoptic Segmentation

Finally, we apply FPG to panoptic segmentation to validate its effectiveness and generalization. By exchanging the FPN part in Panoptic FPN [15] with NAS-FPN or FPG, we compare the results in Table 4. FPG (9@256) is +0.5 higher than NAS-FPN (7@256) and +1.3 higher than FPN in PQ metric. For this task, as we simply adopted all hyper-parameters of Panoptic FPN [15], we expect further gains if these will be tuned in favor of FPG, which is future work.

4.5. Ablation Study

We perform a thorough study of the design of FPG on COCO `val2017`, and then explore different implementations of connections. Ablation experiments are conducted on RetinaNet with the `crop-aug` setting.

Component Analysis. Firstly, we investigate the necessity of pyramid pathways and lateral connections. Starting from a complete version of FPG with all connections and pathways, we remove each component respectively to see the effects. From Table 5 we see that *AcrossDown* is essential for FPG, since it is the only connection that contribute to top-down pathways. Deleting this connection leads to a -4.6 point mAP decrease. On the contrary, *AcrossUp* seems redundant, which only adds to more FLOPs and params but does not improve the performance. Other connections like *SameUp* and *Skip* are beneficial, but with lower impact, as ignoring them results in a -0.5 and -0.6 mAP decrement, respectively. Our grid contraction (Sec. 3.5) which truncates the lower-triangle low-level feature maps (illustrated in Fig. 3) significantly reduces FLOPs and parameters, while maintaining the same performance.

SameUp ($\uparrow$). Table 6 shows the ablation results of the SameUp connection in the pyramid pathway. We compare three commonly used downsampling methods: average pooling, max pooling and 3×3 convolution with a stride of 2. Max pooling outperforms average pooling by +0.6 mAP which is further improved by using Conv (+0.5 mAP).

AcrossDown ($\searrow$). Table 7 shows the ablation results of the across-pathway top-down connections. The simplest implementation is nearest interpolation, as adopted in FPN. Considering that a naïve interpolation may not be enough to build strong top-down pathways, we try attaching a 1×1 or 3×3 convolution to improve the model capacity.

	FLOPs	Params	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
AvgPool	128.07	54.79	0.390	0.584	0.418	0.191	0.444	0.547
MaxPool	128.07	54.79	0.396	0.588	0.426	0.188	0.453	0.559
Conv	136.00	72.50	**0.401**	0.592	0.427	0.194	0.457	0.571

Table 6: Comparison of different designs of *SameUp*.

	FLOPs	#params	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
intp	109.30	57.15	0.319	0.496	0.340	0.141	0.363	0.458
intp + k1	112.30	58.86	0.392	0.580	0.421	0.185	0.446	0.558
intp + k3	136.00	72.50	**0.401**	0.592	0.427	0.194	0.457	0.571

Table 7: Comparison of different designs of *AcrossDown*.

	FLOPs	#param	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
identity	132.99	70.18	0.395	0.581	0.423	0.191	0.453	0.562
k1	136.00	72.50	**0.401**	0.592	0.427	0.194	0.457	0.571

Table 8: Comparison of different designs of *Skip*.

Results show that the accuracy is as low as 31.9 mAP without any convolution layer. Adding an additional 1×1 convolution improves it by +7.3 mAP and adopting a larger kernel size leads to a further +0.9 mAP improvement. Suggesting that a convolutional layer after interpolation, that can adapt the features and re-align the receptive field for further processing, is critical for the implementation of FPG.

Skip ($\curvearrowright$). Skip connection ease the training of deeper pyramids by propagating information across a direct connection. We compare two lightweight designs, an identity connection and 1×1 convolutional projections. As shown in Table 8, 1×1 convolution outperforms identity connection by +0.6 mAP with only marginal extra cost.

5. Conclusion

This paper has presented Feature Pyramid Grids (FPG), a simple multi-pathway extension to FPN, that represents the feature scale-space as a regular grid of parallel pyramid pathways. The pathways are intertwined by multi-directional lateral connections, forming a unified grid of feature pyramids. On detection and segmentation tasks, FPG provides solid improvements over both FPN and NAS-FPN with advantageous accuracy to computation trade-off. Given its simple and intuitive nature, we hope that FPG can serve as a strong baseline for future research and applications in instance-level recognition.

Acknowledgments. We are grateful for discussions with Kaiming He and Ross Girshick.

References

[1] Peter Burt and Edward Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on communications*, 31(4):532–540, 1983. 1, 2

[2] Peter J Burt, Tsai-Hong Hong, and Azriel Rosenfeld. Segmentation and estimation of image region properties through cooperative hierarchial computation. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(12):802–809, 1981. 1, 2

[3] Zhaowei Cai, Quanfu Fan, Rogerio S Feris, and Nuno Vasconcelos. A unified multi-scale deep convolutional neural network for fast object detection. In *Proc. ECCV*. Springer, 2016. 2, 3

[4] Zhaowei Cai and Nuno Vasconcelos. Cascade R-CNN: Delving into high quality object detection. In *Proc. CVPR*, 2018. 2, 5, 6

[5] Yukang Chen, Tong Yang, Xiangyu Zhang, Gaofeng Meng, Chunhong Pan, and Jian Sun. Detnas: Neural architecture search on object detection. *arXiv preprint arXiv:1903.10979*, 2019. 3

[6] Ziteng Gao, Limin Wang, and Gangshan Wu. Lip: Local importance-based pooling. In *Proc. ICCV*, 2019. 5

[7] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. NAS-FPN: Learning scalable feature pyramid architecture for object detection. In *Proc. CVPR*, 2019. 2, 3, 5

[8] Jia Guo, Jiankang Deng, Niannan Xue, and Stefanos Zafeiriou. Stacked dense u-nets with dual transformers for robust face alignment. In *Proc. BMVC.*, 2018. 2

[9] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *Proc. ICCV*, 2017. 2, 5, 6

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. CVPR*, 2016. 3, 4

[11] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 5

[12] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proc. CVPR*, 2018. 5

[13] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q Weinberger. Multi-scale dense networks for resource efficient image classification. *arXiv preprint arXiv:1703.09844*, 2017. 2, 3

[14] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. ICML*, 2015. 5

[15] Alexander Kirillov, Ross Girshick, Kaiming He, and Piotr Dollár. Panoptic feature pyramid networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6399–6408, 2019. 2, 5, 8

[16] Alexander Kirillov, Kaiming He, Ross Girshick, Carsten Rother, and Piotr Dollár. Panoptic segmentation. In *Proc. CVPR*, 2019. 2

[17] Jan J Koenderink. The structure of images. *Biological cybernetics*, 50(5):363–370, 1984. 1, 2

[18] Tao Kong, Fuchun Sun, Anbang Yao, Huaping Liu, Ming Lu, and Yurong Chen. Ron: Reverse connection with objectness prior networks for object detection. In *Proc. CVPR*, 2017. 2, 3

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, 2012. 3, 5

[20] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1989. 1

[21] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proc. CVPR*, 2017. 1, 2, 3, 4

[22] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proc. ICCV*, 2017. 2, 5, 6

[23] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Proc. ECCV*, 2014. 5

[24] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. Autodeeplab: Hierarchical neural architecture search for semantic image segmentation. In *Proc. CVPR*, 2019. 3

[25] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 3

[26] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *Proc. CVPR*, 2018. 2, 3

[27] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *Proc. ECCV*. Springer, 2016. 2, 3

[28] Zili Liu, Tu Zheng, Guodong Xu, Zheng Yang, Haifeng Liu, and Deng Cai. Training-time-friendly network for real-time object detection. *arXiv preprint arXiv:1909.00700*, 2019. 3

[29] Stephane G Mallat. A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (7):674–693, 1989. 1

[30] Junran Peng, Ming Sun, Zhaoxiang Zhang, Tieniu Tan, and Junjie Yan. Efficient neural architecture transformation searchin channel-level for object detection. *arXiv preprint arXiv:1909.02293*, 2019. 3

[31] Pietro Perona and Jitendra Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on pattern analysis and machine intelligence*, 12(7):629–639, 1990. 1, 2

[32] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 2019. 3

[33] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*, 2015. 2, 5, 6

[34] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proc. ICLR*, 2015. 3

[35] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. Deep high-resolution representation learning for human pose estimation. In *Proc. CVPR*, 2019. 3

[36] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proc. CVPR*, 2015. 3

[37] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *Proc. ICML*, 2019. 3

[38] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: Fully convolutional one-stage object detection. In *Proc. ICCV*, 2019. 4

[39] Jiaqi Wang, Kai Chen, Rui Xu, Ziwei Liu, Chen Change Loy, and Dahua Lin. Carafe: Content-aware reassembly of features. In *Proc. ICCV*, 2019. 5

[40] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. `https://github.com/facebookresearch/detectron2`, 2019. 5

[41] Hang Xu, Lewei Yao, Wei Zhang, Xiaodan Liang, and Zhenguo Li. Auto-fpn: Automatic network architecture adaptation for object detection beyond classification. In *Proc. ICCV*, 2019. 2, 3, 5

[42] Fisher Yu, Dequan Wang, Evan Shelhamer, and Trevor Darrell. Deep layer aggregation. In *Proc. CVPR*, 2018. 2, 3

[43] Qijie Zhao, Tao Sheng, Yongtao Wang, Zhi Tang, Ying Chen, Ling Cai, and Haibin Ling. M2det: A single-shot object detector based on multi-level feature pyramid network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 2019. 2, 3