Dynamic Feature Pyramid Networks for Object Detection

Mingjian Zhu[1,3] Kai Han[2] Changbin Yu[3] Yunhe Wang[2]
[1]Zhejiang University. [2]Noah's Ark Lab, Huawei Technologies.
[3]Westlake University.
zhumingjian@zju.edu.cn, {kai.han, yunhe.wang}@huawei.com, yu_lab@westlake.edu.cn

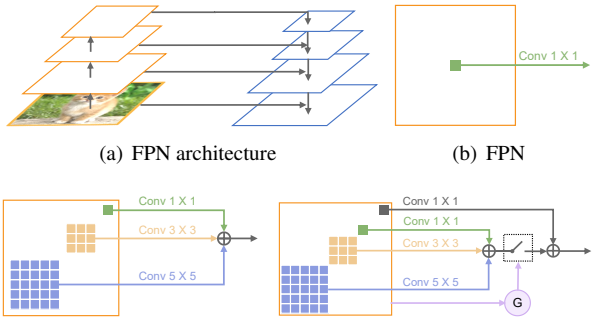
Abstract

This paper studies feature pyramid network (FPN), which is a widely used module for aggregating multi-scale feature information in the object detection system. The performance gain in most of the existing works is mainly contributed to the increase of computation burden, especially the floating number operations (FLOPs). In addition, the multi-scale information within each layer in FPN has not been well investigated. To this end, we first introduce an inception FPN in which each layer contains convolution filters with different kernel sizes to enlarge the receptive field and integrate more useful information. Moreover, we point out that not all objects need such a complicated calculation module and propose a new dynamic FPN (DyFPN). Each layer in the DyFPN consists of multiple branches with different computational costs. Specifically, the output features of DyFPN will be calculated by using the adaptively selected branch according to a learnable gating operation. Therefore, the proposed method can provide a more efficient dynamic inference for achieving a better trade-off between accuracy and detection performance. Extensive experiments conducted on benchmarks demonstrate that the proposed DyFPN significantly improves performance with the optimal allocation of computation resources. For instance, replacing the FPN with the inception FPN improves detection accuracy by 1.6 AP using the Faster R-CNN paradigm on COCO minival, and the DyFPN further reduces about 40% of its FLOPs while maintaining similar performance.

1. Introduction

Object detection is a fundamental task in the computer vision field, which attracts growing attention in recent years. A practical method to detect objects precisely can be useful in modern applications, such as surveillance video and robot navigation. Recent progress in object detection largely stems from the exploitation of deep convolutional neural network (CNN). Devising an effective CNN-based architecture is the mainstream approach for detecting objects across a wide range of scales. The modern detection framework can be categorized as one-stage approach and two-stage approach. The one-stage approaches such as YOLO [24], SSD [21], FCOS [28] and CenterNet [4], directly extract features to predict object classes and locations. In contrast, the two-stage approaches, *e.g.*, Faster R-CNN [26], and Cascade R-CNN [1], firstly obtain the region of interests (ROI) by region proposal network and further generate refined bounding boxes and classes based on ROI. Both approaches make great progress in recent years.

Although considerable object detection approaches have been proposed, the scale variation across object instances still remains a huge challenge. Recently, a number of feature pyramid based techniques are explored to tackle the multi-scale recognition problem and achieved encouraging results. For example, Feature Pyramid Network (FPN) [17] proposes a typical feature pyramid to integrate multi-scale features from the backbone models for object detection.



(a) FPN architecture (b) FPN

(c) inception FPN (d) DyFPN

Figure 1. Different designs of feature pyramid. (a) A feature pyramid is constructed by a bottom-up pathway, a top-down pathway, and lateral connections. (b) At each level, FPN utilizes 1×1 convolution in the lateral connection, which cannot fully explore the multi-scale information. (c) Aggregating the features from different kinds of convolutions generates multi-scale features, which is computationally expensive. (d) Our proposed DyFPN is accurate like (c), but more computationally friendly. The gate in DyFPN determines whether to conduct a combination of convolutions.

The feature pyramid structure in FPN fuses both the deep and shallow feature layers in a top-down manner as shown in Figure 1(a). AugFPN [5] refines the conventional FPN using Consistent Supervision, Residual Feature Augmentation, and Soft RoI Selection, simultaneously. Libra R-CNN [22] strengthens the multi-level features in the feature pyramid by using the integrated balanced semantic features. Although theses methods have made tremendous efforts for improving the performance of feature extractor, the FPN-like framework is mainly focusing on how to aggregate features in different layers. The multi-scale feature information within each convolutional layer is usually ignored. Here, we propose a new method, which not only has the advantages of cross-layer aggregation methods but also efficiently explores the multi-scale information within the layer .

To enhance the representation ability of features generated by FPN, we first embed the inception block into the conventional FPN, *i.e.*, each convolutional layer in our inception FPN contains convolutional filters with different sizes (*e.g.*, 1×1, 3×3, 5×5) as shown in Figure 1(c). Thus, the multi-scale feature maps can be efficiently generated by convolutions with different receptive fields. Compared with the conventional FPN (Figure 1(a) and 1(b)), the inception FPN fully explores the multi-scale information at each level of the feature pyramid, which significantly improves the detection accuracy.

However, the filters with larger sizes obviously increase the overall computational costs. In addition, the scales of objects in different natural images are exactly variant. This observation further motivates us to introduce the dynamic mechanism into the FPN architecture to balance the performance and the overall computation burden. Specifically, a learnable gate block is inserted before a dynamic block in each lateral connection, and the gate block dynamically determines the topology of the dynamic block based on the input. We design two architectures for the dynamic block. In the first architecture, a gate block selects one of the convolutional layers to execute in a dynamic block. In the second architecture, a gate block decides whether to execute the whole dynamic block, as shown in Figure 1(d). We perform experiments to show that both architectures can reduce the computational cost, while the second architecture achieves the best efficiency-accuracy trade-off. Thereby, we consider the second architecture as our final network architecture.

Our main contributions are summarized as follows:

- Different to existing works that investigate the multi-scale features in different layers, this paper excavates the multi-scale features in each layer which allows the output features include more useful information with different receptive fields.

- The effectiveness of DyFPN is well evaluated on various backbone architectures and the MS COCO benchmark. The DyFPN can be easily plugged in the state-of-the-art detection networks to achieve higher performance with slightly higher computational complexity.

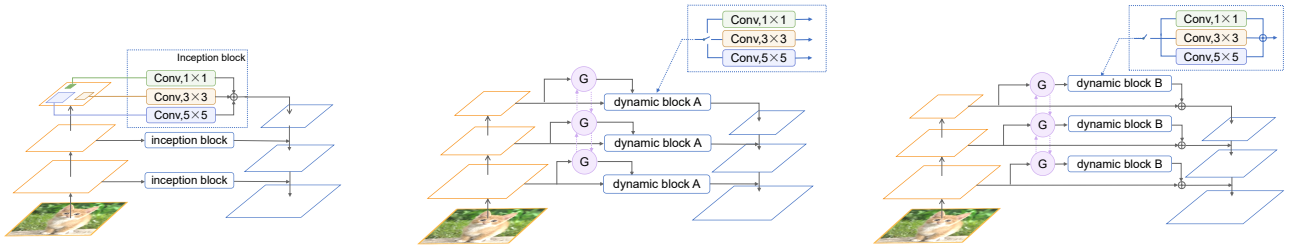
- We demonstrate the effectiveness of DyFPN on MS COCO. Replacing FPN with inception FPN consistently brings significant performance improvements. In addition, DyFPN largely reduces the computational cost of inception FPN while preserving high accuracy.

2. Related Works

Object Detection Objection Detection task aims at recognizing what the object is and where the object locates in an image or a video. Benefit from the deep neural network, many methods on object detection tasks have achieved impressive improvements in recent years. Faster R-CNN [26] proposes an end-to-end detection approach by replacing Selective Search with a novel region proposal network. SSD [21] predicts a series of bounding boxes with different scales and aspect ratios from several feature layers. RetinaNet [18] proposes focal loss to tackle the class imbalance faced by the one-stage detector. Hit-Detector [6] searches for the structure of the backbone, neck, and head altogether in an end-to-end manner. However, the dynamic network has seldom been explored for object detection. In this paper, we propose dynamic FPN to tackle the scale variation problem and further reduce the computational cost.

Deep Feature Pyramids Exploiting features from different scales have been proven effective in many previous works [21, 17, 20, 14]. FPN [17] constructs a bottom-up pathway, a top-down pathway, and lateral connections to fuse the features with different resolutions and scales in an efficient way. PANet [20] further enhances the information propagation ability of FPN with the bottom-up path augmentation. Kong *et al.* [14] combines high-level and low-level features in a nonlinear way with the proposed global and local reconfiguration architecture. CARAFE [30] proposes an effective feature upsampling operators and integrates it into FPN to boost the performance. Panoptic FPN [13] adds the segmentation branch for dense-pixel output to tackle the panoptic segmentation problem. The previous methods basically utilize static architecture, while DyFPN aims at processing features from different levels with dynamic architecture.

Dynamic Neural Networks Previous works on dynamic neural networks mainly focus on adjusting the architecture of models according to the input images [16, 32, 9, 35, 34]. MSDNet [9] adopts dynamic evaluation, which tackles easy examples at early classifiers while handles hard examples with the whole network. S2DNAS [35] proposes a method to transform various CNN models into dynamic models without manually re-designing. For image classification,



(a) inception FPN (b) DyFPN-A (c) DyFPN-B

Figure 2. (a) The inception FPN aggregates the features from multiple convolutions to adapt the receptive fields for objects of different scales. (b) Based on the gate decision, DyFPN-A determines which convolutional layer to conduct in dynamic block A. (c) The gate in DyFPN-B decides whether to execute all the convolutions in dynamic block B. The 1×1 skip-connections are always executed and their outputs are added to the features from dynamic block B. The purple dashed line denotes two kinds of feature input orders (*i.e.*, upward and downward) for the RNN gate block. The content in the dashed box represents the approaches of feature aggregation.

ConvNet-AIG [29] designs gates to determine whether to execute or skip the specific layers, which enables the dynamic adjustment of inference graphs conditioned on the input features. Dynamic routing [16] searches scale transform paths on the fly for semantic segmentation. Different from the previous dynamic networks, we first introduce the inception FPN to improve the accuracy, and then a dynamic structure is introduced to reduce the computational expense.

3. The Proposed Approach

3.1. Inception FPN

By effectively leveraging features from different layers in the backbone model, the feature fusion method is believed to improve the network performance [31, 12, 13, 14, 17]. An efficient method to fuse features is building a feature pyramid. Typically, for a list of input features with different scales, the inception FPN takes a series of features $\{F_2, F_3, F_4, F_5\}$ as input and outputs the aggregated features $\{P_2, P_3, P_4, P_5\}$ as follows:

$$P_5 = f_5(F_5), \tag{1}$$

$$P_l = f_l(F_l) + R(P_{l+1}),\ l = 2, 3, 4, \tag{2}$$

where l denotes the level of the pyramid. R denotes the resizing operation to generate the features that are respectively of the same spatial sizes. The lateral connection $f_l(\cdot)$ is typically a 1×1 convolutional layer which cannot extract multi-scale features in the corresponding level for recognizing scale-variant objects. Thus we propose inception FPN to use inception block as the lateral connection. As shown in Figure 2(a), the inception block consists of a set of convolutions with kernel sizes of 1×1, 3×3 and 5×5 and sums the features from different convolutions as follows:

$$f_l(F_l) = Conv_{1\times1}(F_l) + Conv_{3\times3}(F_l) + Conv_{5\times5}(F_l), \tag{3}$$

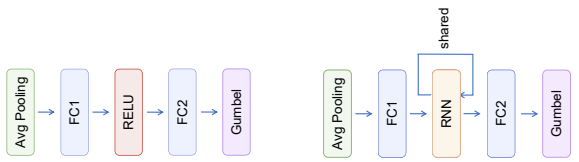
where $l = 2, 3, 4, 5$. With convolutions of different kernel sizes in the lateral connection, the extracted features at each level are multi-scale and have more diverse receptive fields. In the section 4, we conduct experiments to demonstrate that replacing the 1×1, 3×3 and 5×5 convolutions with their dilated variants can further enlarge the receptive fields and achieve better performance.

3.2. Dynamic Feature Pyramid Network

The proposed inception FPN can largely improve the detection accuracy but brings extra computation expenses. Here, we propose DyFPN, which aims at tackling the problems of inception FPN by introducing a novel dynamic mechanism. The overall framework of the DyFPN variants is shown in Figure 2(b) and Figure 2(c). Basically, the DyFPN composes of two components: gate block and dynamic block. We propose two strategies for the gate block to determine the topology of the dynamic block, respectively. First, the gate block selects which one of the convolutional layers to execute in the dynamic block. Second, the convolution layers in the dynamic block can be considered as a whole, and whether to conduct the dynamic block is determined by the gate. In this way, only part of the convolutions in the dynamic blocks are executed for each sample during inference, so the computational cost can be reduced dramatically. We propose two types of the dynamic block (*i.e.* dynamic block A and dynamic block B) and two types of the gate blocks (*i.e.* CNN-based gate and GRU-based gate) to achieve the strategies. The experiments in section 4 demonstrate that the combination of dynamic block B and CNN-based gate achieves the best trade-off between accuracy and computational cost.

3.2.1 Dynamic Block

The architecture variants of DyFPN is illustrated in Figure 2(b) and Figure 2(c). Besides the gate block, DyFPN contains an architecture named dynamic block. In inception FPN, the lateral connections are static. However, the dynamic block determines the operations in lateral connec-



(a) CNNGate (b) GRUGate

Figure 3. Gate block designs. (a) CNNGate contains two fully-connected layers and has an average pooling layer prior to the first fully-connected layer to reduce the computation. (b) GRU gate block designs. Each gate block contains a one-layer GRU with both input and hidden unit size of 16. The parameters of the GRU hidden units are shared across the levels in the feature pyramid.

tions based on the input feature. The dynamic architecture for a specific image can largely reduce the computational cost and retain high detection accuracy. We introduce two dynamic architecture variants as follows, and the performance comparison is provided in section 4.

Dynamic Block A. In Figure 2(b), we demonstrate the details of the dynamic block A. At each level, the gate decides which one of the convolutions to execute conditioned on the input feature. The gate generates a k-d one-hot vector, and k denotes the number of convolution layers in the dynamic block. In the training period, we conduct all convolution layers in the dynamic block and multiply the generated features with the one-hot vector from the gate block. In the testing period, we only execute the convolution layer with the corresponding number equals to 1 in the one-hot vector. Compared with TridentNet [15], which consistently utilizes the single major branch for all the example during inference, our method dynamically selects one of the convolution layers based on the input.

Dynamic Block B. Figure 2(c) shows the details of the dynamic block B. The motivation of the gate block aims at predicting a one-dimensional one-hot vector, which denotes whether to execute or skip the dynamic block B. We sum the features from a set of convolutional layers in dynamic block B. We always conduct a 1×1 convolutional layer at each level and consider it as skip-connection. The features from the dynamic block B and the skip-connection are fused by summation. In the training stage, the prediction of the gate is multiplied with the aggregated features from the dynamic block B. In the testing stage, the dynamic block B does not need to be executed if the gate predicts 0 (*i.e.*, the decision of skipping the computation).

3.2.2 Gate Block

In the gate block, we first apply a non-linear function $g_l(\cdot)$ on F_l and produces the logits of the gate signals:

$$\alpha_l = g_l(F_l), \tag{4}$$

where $a_l \in \mathbb{R}^k$ whose i-th element determines the sampling probability of each lateral connection candidate i at level-l. Here, We set $k = 1$ for the dynamic block A, while k is the number of convolutions for the dynamic block B. Then the one-hot gate vector β_l is obtained by Gumbel Softmax function as follows:

$$\begin{aligned} \beta_l^i &= \textit{GumbelSoftmax}(\alpha_l^i|\alpha_l) \\ &= \frac{exp[\alpha_l^i + n_l^i/\tau]}{\sum_i exp[(\alpha_l^i + n_l^i/\tau)]} \end{aligned} \tag{5}$$

where $\beta_l^i \in \{0, 1\}$, and $n_l^i \sim \textit{Gumbel}(0, 1)$ is a random noise sampled from the Gumbel distribution. τ is a temperature parameter that influences the Gumbel Softmax function. Different implementation of the gate can largely influences the performance of DyFPN. Here, We propose the CNN-based and GRU-based gate blocks and demonstrate their details as follows.

CNN Gate Block. The CNNGate is similar to squeeze-and-excitation (SE) module [8], as shown in Fig. 3(a). The CNNGate consists of a global average pooling layer, two fully-connected layers, a ReLU layer, and a Gumbel Softmax. We reduce the feature dimension by 4 in the first fully-connected layer. The input feature with a shape of (C, H, W) from the selected layer in the backbone model is firstly squeezed by an average pooling operation to produce the feature of size C. Then two fully-connected layers, a non-linear activation function and a Gumbel Softmax function [10, 27, 33]are leveraged to generate the one-hot vector for the dynamic block.

RNN Gate Block. In order to fully exploit the cross-layer dependencies, we employ GRU [3] to determine the topology of the dynamic block, as shown in Figure 3(b). In each selected layer of the backbone model, we take the feature as the input of the GRUGate in the corresponding time step. This process is iterated until all the gate decisions are generated. A global average pooling and a linear projection layer are utilized to project the feature to the input size of 16. Then we utilize the GRU unit with the hidden unit size of 16 to generate output vectors. For each gate output, we apply linear projection and Gumbel Softmax function to obtain a one-hot vector. The GRUGate enables parameter sharing and allows the reuse of computation across layers in the feature pyramid. We compare the CNN-based and GRU-based gate block in section 4.

3.3. Resource Constraint

In real-world scenarios, physical devices impose different computing resource constraints on the model. DyFPN can be trained with the consideration of computational expenses. Here, we denote C as the computational cost of the model, *i.e.*, FLOPs. The maximum and minimum costs of the dynamic block can be calculated before training the

model and we denote them as C_{max} and C_{min}. For DyFPN-A, the maximum cost appears when the gate consistently selects 5×5 convolutional layer in each lateral connection. When all the gates choose 1×1 convolutional kernel, the model consumes minimum computational cost. For DyFPN-B, the decision of executing or skipping the dynamic block in all lateral connections of the feature pyramid leads to maximum or minimum computational cost, respectively. Here we introduce the losses for the end-to-end optimization:

$$\mathcal{L}_C = ((C_R - C_{target})/(C_{max} - C_{min}))^2 \quad (6)$$

where C_R denotes the real computational cost of the dynamic block. C_{target} represents the target resource cost. We can control the target cost by setting the hyper-parameter $\alpha \in (0, 1)$. The target cost can be formulated as follows:

$$C_{target} = C_{min} + \alpha * (C_{max} - C_{min}) \quad (7)$$

The total loss function can be optimized as follows:

$$\mathcal{L} = \mathcal{L}_{Det} + \lambda \mathcal{L}_C \quad (8)$$

where $\mathcal{L}_{Det}$ and $\mathcal{L}_C$ represent the loss of detection and computational cost, respectively. We leverage λ to balance the detection accuracy expectation and computational cost constraints, respectively.

4. Experiments

4.1. Dataset and Evaluation Metrics

All experiments are conducted on the MS COCO 2017 detection dataset [19], which contains 80 object categories. Following the protocol in MS COCO, 118k images are used for training. We report the results of ablation studies for *minival* with 5k validation images. To compare with the state-of-art methods, we report model performances on the *test-dev* set, which requires the prediction results uploaded to the evaluation server. Following common practice, COCO Average Precision(AP) with different IoU thresholds is leveraged as the evaluation metric [6, 36].

4.2. Implementation Details

For all experiments, our detectors are trained end-to-end on a machine with 4 NVIDIA RTX 2080Ti GPUs. We utilize the SGD optimizer to train the model for 12 epochs, known as $1\times$ schedule. Linear warm-up strategy for 500 iterations is leveraged at the beginning of training. We initialize the learning rate as 0.01 and decrease to 0.001 and 0.0001 at 8th-epoch and 11th-epoch. The momentum is set as 0.9 and the weight decay is 0.0001. The batch size is set to 2 per GPU. The input size is 1333×800. The factor τ in Eq. 5 is set to 1.0. Our implementation is developed by PyTorch framework [23] and mmdetection toolbox [2].

4.3. Main Results

Effectiveness and Efficiency of DyFPN.
To demonstrate the significance of our method, we compare DyFPN with model variants of FPN and inception FPN in Figure 4. We can see that DyFPN achieves the best efficiency-accuracy trade-off among the model variants. In each lateral connection of FPN, we replace 1×1 convolutional layer with 3×3, 5×5 and 7×7 convolutional layer, respectively. The FPN with a 7×7 convolutional layer achieves the best performance while consumes most computational costs. The model with lowest accuracy follows the architecture of the conventional FPN. It can be observed that enlarging the size of convolutional kernel in FPN improves the accuracy while consumes more computational costs, which is not efficient. We combine different kinds of the convolutional layers in the inception block, which accounts for different computational costs and accuracies. Although simply adding more convolutional layers can improve the detection accuracy, the consequent computational burden can not be neglected. Our DyFPN always outperforms the FPN and inception FPN with similar computational costs. From the comparisons, we can conclude that our proposed dynamic mechanism can improve the performance of FPN in an efficient manner.

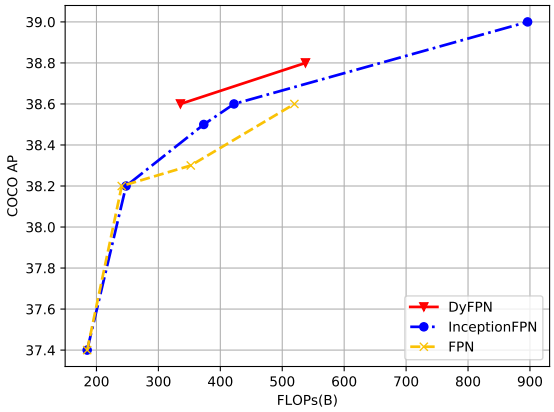


Figure 4. Accuracy *v.s.* FLOPs on COCO *minival*.

Comparison with State-of-the-Arts.
In Table 1, we compare the results of our DyFPN with the other state-of-the-art detectors on COCO test-dev set. To demonstrate the superiority of our method, we summarize the results of the classical architectures and recent state-of-the-art feature pyramids. For a fair comparison, most of the detectors are based on Faster R-CNN [26] and FPN [17]. Without bells and whistles, DyFPN achieves 40.8 AP with ResNet-101 backbone in the $1\times$ training scheme. Considering that DyFPN is an augmented version of FPN, we compare its performance with the other detectors, which focus on improving the accuracy of FPN.

Table 1. Comparison of single-model results on COCO test-dev. The symbol '*' denotes the results from our re-implementation.

Method	Backbone	mAP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
YoLOv3 [25]	Darknet-53	33.0	57.9	34.4	18.3	25.4	41.9
SSD512 [21]	ResNet-101	31.2	50.4	33.3	10.2	34.5	49.8
RetinaNet [18]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
Faster R-CNN [26]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Mask R-CNN [7]	ResNet-101-FPN	38.2	60.3	41.7	20.1	41.1	50.2
IoU-Net [11]	ResNet-101-FPN	40.6	59.0	55.2	49.0	38.0	17.1
CARAFE [30]	ResNet-50-Faster R-CNN	38.1	60.7	41.0	22.8	41.2	46.9
Libra R-CNN [22]	ResNet-50-Faster R-CNN	38.7	59.9	42.0	22.5	41.1	48.8
Libra R-CNN [22]	ResNet-101-Faster R-CNN	40.3	61.3	43.9	22.9	43.1	51.0
AugFPN [5]	ResNet-50-Faster R-CNN	38.8	61.5	42.0	23.3	42.1	47.7
AugFPN [5]	ResNet-101-Faster R-CNN	40.6	63.2	44.0	24.0	44.1	51.0
FPN*	ResNet-50-Faster R-CNN	37.7	58.7	40.8	21.7	40.6	46.7
FPN*	ResNet-101-Faster R-CNN	39.7	60.7	43.2	22.5	42.9	49.9
inception FPN	ResNet-50-Faster R-CNN	39.2	60.9	42.5	22.8	42.0	48.9
inception FPN	ResNet-101-Faster R-CNN	40.9	62.4	44.6	23.6	44.1	51.7
DyFPN	ResNet-50-Faster R-CNN	39.0	60.7	42.2	22.4	41.8	49.0
DyFPN	ResNet-101-Faster R-CNN	40.8	62.4	44.6	23.4	44.2	51.7

In particular, DyFPN outperforms CARAFE [30], Libra R-CNN [22], and AugFPN [5] with the same frameworks and backbones. To fairly compare with the baseline method, we re-implement FPN with different backbones. We can see that DyFPN consistently outperforms FPN with a large margin. When using ResNet-50 and ResNet-101 as the backbones, replacing FPN with inception FPN improves the Faster R-CNN by 1.5 and 1.2 AP, respectively. The DyFPN performs similar to inception FPN, which achieves 1.3 and 1.1 AP better than FPN.

4.4. Ablation Study

In this section, we perform experiments on COCO *mini-val* to demonstrate the effectiveness of our method. All the experiments are conducted on Faster R-CNN framework with ResNet-50 as the backbone unless specified.

Components of the Inception Block.

In Table 2, We conduct experiments to demonstrate that different combinations of the convolutional layers in the inception block achieve distinct performance. We first follow the architecture of FPN [17] and demonstrate its performance in Conv1 (d=1), where d denotes the dilation rate. Based on FPN, we add 3×3 convolution in each lateral connection, which leads to 0.8% absolute gain in AP. We can see that the addition of 3×3 convolution brings additional improvements for all sizes of objects. We further add 5×5 convolution and find 0.4% increase in AP. However, the addition of 5×5 convolution achieves improvement for middle and large objects but drops in performance for small objects. We conjecture that the receptive field of 5×5 convolution overemphasizes large objects, which drowns the information of the small objects in the aggregated features. We add some convolutions with different dilation rates to further increase the receptive field. Note that although the 3×3 convolution with dilation rate 2 has the same receptive field as the 5×5 convolution, their parameters and computations are not the same. Thus, employing Conv3 (d=2) or Conv5 (d=1) leads to different results in the inception block. It can be observed that the addition of convolutions with a dilation rate of 2 and 3 improves the detection accuracy in almost all scales. Aggregating features from convolutions of different kernel sizes and dilation rates enables the receptive fields of the detection model to cover objects with different scales, which results in accuracy improvement. To achieve the best accuracy, the convolutional layers in our inception block finally adopt the following configurations: kernel size = [1, 3, 3, 3, 5, 5, 5], dilation rate = [1, 1, 2, 3, 1, 2, 3], padding = [0, 1, 2, 3, 2, 4, 6].

Extension to Different Frameworks.

We compare the inception FPN with the baseline FPN on different frameworks to demonstrate the effectiveness of the inception FPN. In Figure 5, we can see that the inception FPN outperforms the baseline on all frameworks. Based on Faster R-CNN, the inception FPN (39.0%) achieves 1.6 AP better than FPN (37.4%) with Resnet-50 as the backbone. Basically, the methods based on Cascade R-CNN [1] are better than that based on Faster R-CNN. The inception FPN achieves the best results (42.6%) with Resnet-101 as the backbone. The inception FPN performs 1.0 and 0.6 AP better than FPN with Resnet-50 and Resnet-101, respectively. In terms of Mask R-CNN [7], the inception FPN improves over the FPN with 1.3 and 1.0 AP increase. We can see that the introduced inception block can always boost the detection accuracy for different frameworks.

Performance of Model Variants in DyFPN.

To compare the performance of model variants, we apply

Table 2. Alation study of inception FPN. The detection results are evaluated on COCO *minival*.

Conv3 (d=1)	Conv3 (d=1)	Conv5 (d=1)	Conv3 (d=2)	Conv3 (d=3)	Conv5 (d=2)	Conv5 (d=3)	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
✓							37.4	58.3	40.4	21.2	41.1	48.6
✓	✓						38.2	59.2	41.1	22.1	41.8	49.4
✓	✓	✓					38.6	59.9	41.6	21.9	42.1	50.4
✓	✓		✓	✓			38.5	59.9	41.8	22.0	42.2	50.1
✓	✓	✓	✓	✓	✓	✓	**39.0**	**60.6**	**42.4**	**23.3**	**42.7**	**50.5**

Table 3. Evaluations of different designs of DyFPN. The models are trained for 12 epochs.

DyFPN	Gate Block	#FLOPs(B)	#params	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
A	GRU(upward)	242.0	142.1	37.8	58.8	41.0	21.4	41.3	49.0
	GRU(downward)	324.5	142.1	38.3	59.5	41.3	22.5	41.9	49.8
	CNN	264.6	143.4	37.9	58.9	40.9	21.9	41.7	49.1
B	GRU(upward)	546.2	143.1	38.7	60.3	42.0	22.6	42.1	50.7
	GRU(downward)	555.9	143.1	38.6	60.4	41.5	22.6	42.6	50.3
	CNN	**537.5**	**144.4**	**38.8**	**60.5**	**42.0**	**22.9**	**42.5**	**50.0**

different gates on DyFPN-A and DyFPN-B in Table 3. Considering the data-dependent property of the dynamic feature pyramid network, we report the average FLOPs in the COCO *minival*. On DyFPN-A, the gate determines which convolutional layer to execute in dynamic block A. On DyFPN-B, the gate decides whether to conduct the dynamic block B. Both DyFPN-A and DyFPN-B employ 7 kinds of convolutional layers in Table 2. Two kinds of gates are utilized to construct an efficiency-accuracy trade-off model, *i.e*. GRUGate and CNNGate. In each time step, We input one feature to the GRUGate. According to the order of input features, the GRUGate can be categorized as *upward* and *downward*. For a list of input features $F = (F_2, F_3, F_4, F_5)$ from level 2-5 in backbone model, *upward* method inputs features to the GRUGate following the order of $G_{upward} = (F_2, F_3, F_4, F_5)$, while *downward* method follows $G_{downward} = (F_5, F_4, F_3, F_2)$. We also demonstrate the performance of the CNNGate. The output of each gate controls the dynamic block at the same level of the feature pyramid. It can be seen that employing CNNGate in the DyFPN-B achieves the best acccuracy. We consider this architecture as our final DyFPN architecture. In the following, DyFPN adopts this architecture unless specified.

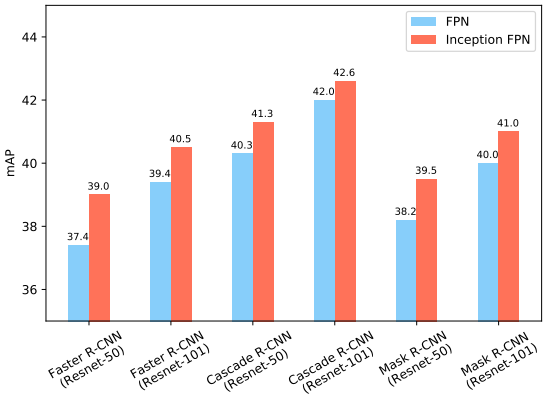


Figure 5. The comparison between FPN and inception FPN on different framework and backbones.

Table 4. Comparison of inception FPN and DyFPN.

Model	#FLOPs(B)	AP	AP_{50}	AP_{75}
inception FPN	896.2	39.0	60.6	42.4
DyFPN	537.5	38.8	60.5	42.0

In Table 4 we compare DyFPN with inception FPN. Our DyFPN reduces 40.0% computational cost in FLOPs with only 0.2 degradation in AP. From the results, we can draw a conclusion that our dynamic mechanism can largely reduce the amount of computation, while preserves high accuracy.

Effectiveness of the Gate Block.

To demonstrate the effectiveness of our proposed gate block, we compare the performance of DyFPN with signals from the learnable gates and the random number generator. In Table 5, we apply different signals to DyFPN-A and DyFPN-B in the training and testing stage. Here, we compare the random signals with the signals from the GRUGate and CNNGate. we utilize GRU(downward) in DyFPN-A and CNNGate in DyFPN-B. The random signals are randomly generated one-hot vectors. In DyFPN-A, we randomly select convolutional layers in dynamic block A. In DyFPN-B, we randomly decide whether to execute the convolutional layers in dynamic block B. It can be observed that replacing the gate signals with the randomly generated signals in the testing period always leads to a dramatic performance drop, which indicates that the gate signals are significant. For a fair comparison, we test the model 10 times with the random signals and average their results. Consider-

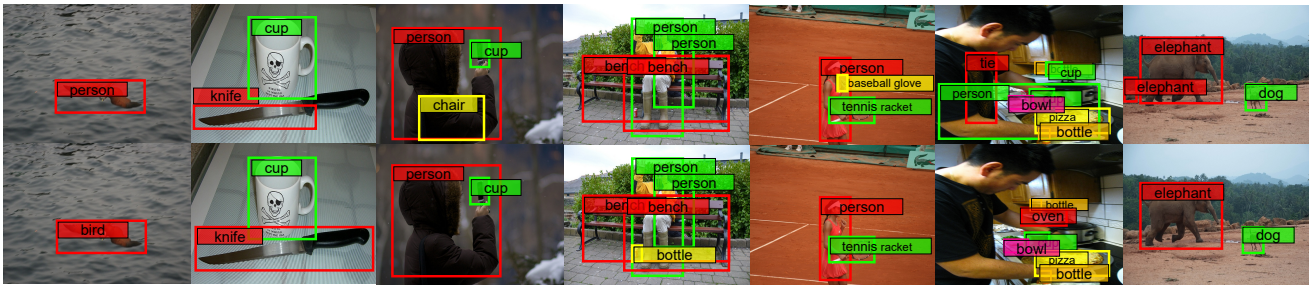


Figure 6. Qualitative results comparison. The results of FPN are listed in the first row, while those of DyFPN are in the second row.

ing that the inconsistency between train/test-time inference leads to performance degradation to the model, we train the models with the random signals for a fair comparison. We can see that training the DyFPN with the random signals makes the model robust to the disturbance of the stochastic factors in testing. However, our gate signals still outperform the random signals with a large margin.

Table 5. Ablation studies of different kind of signals. *R* denotes the random signals. *G* is the GRUGate. *C* means the CNNGate.

DyFPN	Train	Test	AP	AP_{50}	AP_{75}
A	*R*	*R*	36.1	56.7	39.1
	G	*R*	6.5	10.3	6.8
	G	*G*	38.3	59.5	41.3
B	*R*	*R*	36.9	57.7	39.9
	C	*R*	18.8	30.0	19.8
	C	*C*	38.8	60.5	42.0

Computational Resource Budgets.
In Section 3.3, we propose a computational cost loss to restrict the computational cost of the model. In Table 6, we leverage CNNGate and dynamic block B to construct DyFPN and train it with different settings. By tuning the hyper-parameter α and λ, we can train DyFPN-I, DyFPN-II, DyFPN-III, and DyFPN-IV with different resources constraints. We compare the resource constraint models with DyFPN-Raw, which is trained without the usage of computational loss. By tuning α, we can set different target computational cost for DyFPN. It can be seen that the real computational costs of DyFPN can descend to different levels while we employ different targets. With the strongest constraint, the DyFPN-IV is decreased to 61.1% compared with DyFPN-Raw. In addition, the computational loss enables DyFPN-I to consume fewer computational costs while preserving similar accuracy.

In Figure 6, we demonstrate some detection examples on COCO *minival* to qualitatively compare the differences between FPN and DyFPN.It can be seen that our DyFPN generates satisfactory results for small, medium, and large objects by exploring richer receptive fields, while the conventional FPN generates inferior results. In Figure 7, we visualize the features generated by convolutions in inception FPN. We can see that simply exploring the multi-scale information in different levels of the feature pyramid cannot obtain satisfactory results while utilizing the convolutions with different kernel sizes can generate multi-scale features for detection.

Table 6. Ablation studies of different resource budgets. λ and α denote the coefficients for the budget constraint.

Model	α	λ	AP	#FLOPs(B)
DyFPN-Raw	0.0	0.0	38.8	537.5
DyFPN-I	0.5	0.1	38.7	523.6
DyFPN-II	0.3	0.1	38.6	399.3
DyFPN-III	0.2	0.1	38.4	329.0
DyFPN-IV	0.2	0.5	38.3	328.6

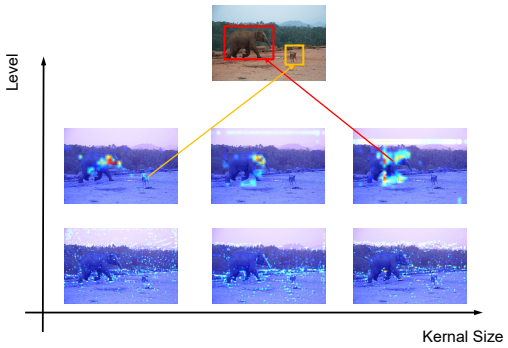


Figure 7. The qualitative results of the features.

5. Conclusion

In this work, we propose a novel dynamic FPN for object detection, in which the gate block adjusts the topology of the feature pyramid to the input feature. We first introduce an inception FPN to improve the accuracy, and then the dynamic FPN is further proposed to reduce the computational cost while preserving accuracy. Given input features, the gate block determines whether to execute the combinations of the convolutional layers in DyFPN. The dynamic pyramid network can efficiently and effectively explore multi-scale information at each level of the feature pyramid. Experiments on the MS COCO dataset demonstrate the effectiveness of the proposed DyFPN.

References

[1] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6154–6162, 2018. 1, 6

[2] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019. 5

[3] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014. 4

[4] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. Centernet: Keypoint triplets for object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6569–6578, 2019. 1

[5] Chaoxu Guo, Bin Fan, Qian Zhang, Shiming Xiang, and Chunhong Pan. Augfpn: Improving multi-scale feature learning for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12595–12604, 2020. 2, 6

[6] Jianyuan Guo, Kai Han, Yunhe Wang, Chao Zhang, Zhaohui Yang, Han Wu, Xinghao Chen, and Chang Xu. Hit-detector: Hierarchical trinity architecture search for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11405–11414, 2020. 2, 5

[7] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017. 6

[8] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. 2018. 4

[9] 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

[10] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *ArXiv*, abs/1611.01144, 2017. 4

[11] Borui Jiang, Ruixuan Luo, Jiayuan Mao, Tete Xiao, and Yuning Jiang. Acquisition of localization confidence for accurate object detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–799, 2018. 6

[12] Seung-Wook Kim, Hyong-Keun Kook, Jee-Young Sun, Mun-Cheon Kang, and Sung-Jea Ko. Parallel feature pyramid network for object detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 234–250, 2018. 3

[13] 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, 3

[14] Tao Kong, Fuchun Sun, Chuanqi Tan, Huaping Liu, and Wenbing Huang. Deep feature pyramid reconfiguration for object detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 169–185, 2018. 2, 3

[15] Yanghao Li, Yuntao Chen, Naiyan Wang, and Zhaoxiang Zhang. Scale-aware trident networks for object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 6054–6063, 2019. 4

[16] Yanwei Li, Lin Song, Yukang Chen, Zeming Li, Xiangyu Zhang, Xingang Wang, and Jian Sun. Learning dynamic routing for semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8553–8562, 2020. 2, 3

[17] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017. 1, 2, 3, 5, 6

[18] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017. 2, 6

[19] Tsung-Yi Lin, M. Maire, Serge J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. *ArXiv*, abs/1405.0312, 2014. 5

[20] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8759–8768, 2018. 2

[21] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016. 1, 2, 6

[22] Jiangmiao Pang, Kai Chen, Jianping Shi, Huajun Feng, Wanli Ouyang, and Dahua Lin. Libra r-cnn: Towards balanced learning for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 821–830, 2019. 2, 6

[23] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019. 5

[24] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. 1

[25] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. 6

[26] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015. 1, 2, 5, 6

[27] Christian Szegedy, W. Liu, Y. Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, D. Erhan, V. Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015. 4

[28] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: Fully convolutional one-stage object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 9627–9636, 2019. 1

[29] Andreas Veit and Serge Belongie. Convolutional networks with adaptive inference graphs. 2018. 3

[30] Jiaqi Wang, Kai Chen, Rui Xu, Ziwei Liu, Chen Change Loy, and Dahua Lin. Carafe: Content-aware reassembly of features. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3007–3016, 2019. 2, 6

[31] Tiancai Wang, Rao Muhammad Anwer, Hisham Cholakkal, Fahad Shahbaz Khan, Yanwei Pang, and Ling Shao. Learning rich features at high-speed for single-shot object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1971–1980, 2019. 3

[32] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 409–424, 2018. 2

[33] B. Wu, Xiaoliang Dai, P. Zhang, Y. Wang, Fei Sun, Yiming Wu, Yuandong Tian, P. Vajda, Y. Jia, and K. Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10726–10734, 2019. 4

[34] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8817–8826, 2018. 2

[35] Zhihang Yuan, Bingzhe Wu, Zheng Liang, Shiwan Zhao, Weichen Bi, and Guangyu Sun. S2dnas: Transforming static cnn model for dynamic inference via neural architecture search. *arXiv preprint arXiv:1911.07033*, 2019. 2

[36] 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, pages 9259–9266, 2019. 5