# *Supplementary material for* Scale-equalizing Pyramid Convolution for object detection

Xinjiang Wang[*], Shilong Zhang[*], Zhuoran Yu, Litong Feng, Wayne Zhang
Sensetime Research
{wangxinjiang, zhangshilong, yuzhuoran, fenglitong, wayne.zhang}@sensetime.com

## Abstract

*We present details of FLOPS calculation in Sec. 1. The pseudo-code of pyramid convolution is in Sec. 2. The detailed mathematical proof that pyramid convolution can extract scale-invariant features from Gaussian pyramid is in Sec. 3. Sec. 4 shows the details of experiments including ablation study and test of inference time respectively. Extra ablation experiments of integrated batch normalization(iBN) and the effect of the number of PConv layers are presented in Sec. 5. Finally, We present implementation details of Feature Selective Anchor Free module in Sec. 6.*

## 1. FLOPs in the head

The input image size of all models is $3 \times 1280 \times 800$. we follow the computation of FLOPs in mmdetection[1] for normal convolution, where one multiplication-addition pair is counted for a single operation. This yields

$$\text{FLOPs} = C_{in} \times K_h \times K_w \times H \times W \times C_{out} \tag{1}$$

for a single convolution operation, where $C_{in}$ and $C_{out}$ are the number of input and output channels of a convolution, which are all fixed as 256 in this study, $K_h$ and $K_w$ are the convolutional kernel sizes and set as 3, $H$ and $W$ are the width and height of a feature map.

PConv fuses features of adjacent levels to the work-on level by 3-D convolutions with stride of 0.5(for upper level) and of 2(for lower level). For the convolution with stride of 0.5, we implemented by performing a regular convolution with stride of 1 followed by an upsample operation, which can be represented in formula as:

$$y^l = \text{Upsample}(w_1 * x^{l+1}) + w_0 * x^l + w_{-1} *_{s2} x^{l-1}, \tag{2}$$

where $x^l$ is the feature map at level $l$ and $w_1, w_0, w_{-1}$ are three independent convolutional kernels. Since the feature map size of $x^{l+1}$ is half of $x^l$ and the size of $x^{l-1}$ is twice as much as $x^l$, the computational cost of the first term in Eqn.2 is a quarter of that of the second term whereas the last two terms share the same computational cost. Note that at top-most level(P7), the first term is eliminated and at bottom-most level(P3), the last term is eliminated. We abbreviate the analysis of upsample operation as its cost is relatively small compared to convolutions in this place. Therefore, the ratio $c_i$ of the FLOPs after applying PConv to that of the original ones in feature pyramids are 2, 2.25, 2.25, 2.25, 1.25 in top-down order. Additionally, the FLOPS associated with each level is also proportional to $H \times W$. The ratio between the spatial size of each feature map and total spatial size can be represented by $r_i = \frac{H_i \times W_i}{\sum_{j=3}^{7} H_j \times W_j}$. The numeric values of such ratios are calculated as: 0.0029,0.0117,0.0469,0.1877,0.7507 in top-down order. Thus, the total computation of using 4 stacked PConv is $C = \sum_{i=3}^{7} c_i \times r_i = 1.4985$ times of using 4 stakced convolution in regular settings.

When classification and regression subsets use combined PConv structure with one extra non-combined for each subnet, the total computation is $\frac{(4 \times C + 2 \times 1)}{2 \times 4 \times 1} = \mathbf{0.99925}$ times of that using default head design.

When SEPC or SEPC-Lite is used, we discuss how FLOPs in deformable conv are calculated. One forwarding of deformable conv is composed of a normal convolution with output channels of $2 \times K_h \times K_w$ for offset prediction, a bilinear

[*]equal contribution

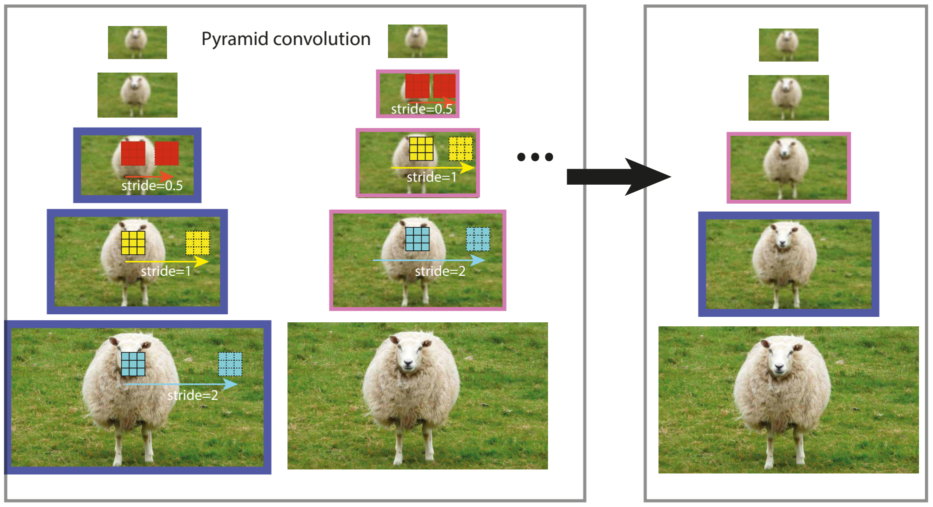

Figure 1: Illustration of pyramid convolution on the feature pyramid

interpolation for every sampling point which involves 8 matiplications and 7 additions, and another normal convolution. So we get

$$FLOPS \approx (1 + \frac{8 + 2 \times K_h \times K_w}{C_{out}}) \times C_{in} \times K_h \times K_w \times H \times W \times C_{out}$$

In SEPC-Lite, we use a normal conv2D on P3 and deform it through P4-P7. Notice that $\frac{\sum_{i=4}^{7} H_i \times W_i}{\sum_{i=3}^{7} H_i \times W_i} \approx 0.249$, using each deformable convolution only introduces $0.249 \times \frac{26}{256} = 0.025$ times more computation comparted to using normal conv in feature pyramid. When it comes to SEPC, the actual method for evaluating computation cost is slightly different from calculation in SEPC-Lite due to P3 and P7, however, following a similar trajectory, one can easily justify that the extra computation cost is still marginal.

## 2. Implementation pseudocode of PConv and SEPC

The pseudocodes of pyramid conv (PConv) are attached as follows, which also corresponds to Fig. 3 & 4(a) in the manuscript. Note that only normal Conv2D modules are used in pconv.

```
def pconv_module_forward(x, conv2D_list):
  # x: input feature list [p3,p4,p5,p6,p7]
  # conv2D_list: conv2D module list,
  # [nn.Conv2D(stride=2),nn.Conv2D(),nn.Conv2D()]
  out_x = []
  for level in range(len(x)):
    tmp = conv2D_list[1](x[level])
    if level > 0:
      tmp += conv2D_list[0](x[level-1])
    if level < len(x) - 1:
      tmp += Upsample(conv2D_list[2](x[level+1])
    out_x.append(tmp)
  return out_x
```

As for Scale-equalizing pyramid conv (SEPC), within the `for` loop of the pseudocodes, `conv2d_list[i]` changes to `DeformableConv(conv2d_list[i].weight)`, where `i` $\in \{0, 1, 2\}$, **only when** `level > 0` (i.e. excluding P3 layer). The main idea is illustrated in Fig. 6 in the manuscript. Note that when convolving the lowest-level features (P3), a normal Conv2D is utilized, whose weights are shared by deformable convs used in higher layers.

Therefore, SEPC is an improved version of pconv, to relax the discrepancy of feature pyramid from a Gaussian pyramid by aligning the feature map of higher layers with the lowest layer. In one word, pconv only uses plain Conv2D modules while SEPC leverages deformable convs in an efficient way. Different from a naive implementation of deformable convolution in the head, SEPC only deforms the kernel when convolving higher layers, which is motivated by the perspective from scale space theory and is much more effective w.r.t. computational cost gain.

## 3. Discussion about remark 1

**Remark 1.** *Pyramid convolution is able to extract scale-invariant features from a Gaussian pyramid.*

**Gaussian scale space.** Consider an image $f : \mathbb{Z}^2 \to \mathbb{R}^2$, where the input domain represents the pixel coordinate and $f(\mathbf{x})$ is pixel intensity, a Gaussian scale space is generated by consecutively blurring the initial image $f_0$ with an isotropic 2-D Gauss-Weierstrass kernel $G(\mathbf{x}, t) = (4\pi t)^{-1}\exp(\|\mathbf{x}\|^2/4t)$ of variable width $\sqrt{t}$ and spatial position $\mathbf{x}$. A set of responses $f(t, x)$, $t \geq 0$ represents blurred images, forming a Gaussian scale space [3] (GSS), as written by:

$$f(t, \mathbf{x}) = [G(\cdot, t) * f_0](\mathbf{x}), \quad t > 0 \tag{3}$$

where a higher $t$ indicates a larger blur.

**Gaussian pyramid.** With the above introduction of GSS, a Gaussian pyramid is denoted as

$$p(a, \mathbf{x}) = f(t(a, s_0), a^{-1}\mathbf{x}) \tag{4}$$

where $s_0$ is the initial scale, $a$ is the downsizing ratio $0 < a \leq 1$ and

$$t = \frac{s_0}{a^2} - s_0 \tag{5}$$

is the Gaussian kernel variance corresponding to the downsizing ratio $a$ in order to keep the same frequency limit after downsizing[4]. In practice, $a$ is chosen to be $2^{-l}$, where $l$ is the level of Gaussian pyramid with $l = 0$ denoting no sub-sampling on the original image. Then the Gaussian pyramid is also written as

$$p_l(\mathbf{x}) = f(t(2^{-l}, s_0), 2^l\mathbf{x}). \tag{6}$$

In fact, we can also define an action $S_n$ that transfer from the original level into another by,

$$[S_n[p]](\mathbf{x}) = p_n(\mathbf{x}) = [G(\cdot, t(2^{-n}, s_0)) * p_0](2^n\mathbf{x}). \tag{7}$$

**Lemma 1.** *The actions $S_m$ and $S_n$ satisfy $S_mS_n = S_{m+n}$*

*Proof.* Since the sub-sampling ratio $2^m2^n = 2^{m+n}$ naturally satisfies this argument, we mainly focus on the Gaussian convolution in Eqn. 7. In order to prove the associativity property of $S_n$ and $S_m$, we only need to calculate the associativity property of convolution

$$(G(\cdot, t(2^{-m}, s_0)) * [G(\cdot, t(2^{-n}, s_0)) * f(\cdot)](2^n\mathbf{x}))(2^m\mathbf{x}) = [G(\cdot, t(2^{-m-n}, s_0)) * f(\cdot)](2^{m+n}\mathbf{x}) \tag{8}$$

This process contains four sub-processes: (1) Gaussian convolution with variance $t(2^{-n})$, (2) sub-sampling by $2^n$; (3) Gaussian convolution with variance $t(2^{-m})$; (4) sub-sampling by $2^m$. Then we will follow this process and apply Fourier transform on after another. Applying Fourier transform to $G(\cdot, t(2^{-n}, s_0)) * f(\cdot)$, we obtain

$$\mathcal{F}_1 = \mathcal{F}\left([G(\cdot, t(2^{-n}, s_0) * f(\cdot)])\right) = \exp(-t(2^{-n}, s_0)\|\omega\|^2))\mathcal{F}_0(\omega), \tag{9}$$

where $\omega \in \mathbb{R}^2$ represents 2-D frequency in the Fourier-transformed domain, $\mathcal{F}_0(\omega)$ is the Fourier transform of $f(\mathbf{x})$. Substituting Eqn. 5 into the above equation acquires

$$\mathcal{F}_1 = \exp((2^{2n}s_0 - s_0)\|\omega\|^2))\mathcal{F}_0(\omega), \tag{10}$$

After process (2) (sub-sampling by $2^n$), the Fourier transform is now

$$\mathcal{F}_2 = \mathcal{F}\left([G(\cdot, t(2^{-n}, s_0) * f(\cdot)])(2^n\mathbf{x})\right) = 2^{-n}\exp(-t(2^{-n}, s_0)\left\|2^{-n}\omega\right\|^2))\mathcal{F}_0(2^{-n}\omega) = 2^{-n}\exp((s_0 - 2^{-2n}s_0)\|\omega\|^2))\mathcal{F}_0(2^{-n}\omega), \tag{11}$$

As for process (3) (Gaussian convolution with variance $t(2^{-m})$),

$$\mathcal{F}_3 = \mathcal{F}\left([G(\cdot, t(2^{-m}, s_0)])\right) \cdot \mathcal{F}_2 = 2^{-n}\exp((2^{2m}s_0 - 2^{-2n}s_0)\|\omega\|^2))\mathcal{F}_0(2^{-n}\omega), \tag{12}$$

As for process (4) (sub-sampling by $2^{-m}$),

$$\mathcal{F}_4 = 2^{-m-n}\exp((s_0 - 2^{-2n-2m}s_0)\|\omega\|^2))\mathcal{F}_0(2^{-m-n}\omega), \tag{13}$$

It is obvious that after process 4), the Fourier-transformed expression is equivalent to the Fourier transform of RHS of Eqn. 8

$$\mathcal{F}\left([G(\cdot, t(2^{-m-n}, s_0) * f(\cdot)])(2^{m+n}\mathbf{x})\right) = 2^{-m-n}\exp((s_0 - 2^{-2n-2m}s_0)\|\omega\|^2))\mathcal{F}_0(2^{-m-n}\omega) = \mathcal{F}_4 \tag{14}$$

□

The above lemma is very useful in a Gaussian pyramid, since it means that one level in the Gaussian pyramid is able to be transferred to another by a simple *jumping* action, such that

$$p_{m+n} = S_m[p_n] \tag{15}$$

Now recall that the expression of pyramid convolution is given by

$$y^l = w_1 *_{s0.5} x^{l+1} + w_0 * x^l + w_{-1} *_{s2} x^{l-1}, \tag{16}$$

where $w_1$, $w_0$ and $w_{-1}$ are three independent kernels, $*_2$ denotes a convolution with stride 2, $x^l$ represents the feature pyramid in the $l^{th}$ layer. Once the feature pyramid $x^l$ can be viewed as a Gaussian pyramid, the pyramid convolution is written as

$$y^l(\mathbf{z}) = \sum_{k=-1}^{1} [w_k * p_{l+k}](2^{-k}\mathbf{z}) = \sum_{k=-1}^{1} \sum_{\mathbf{u}\in\mathbb{Z}^d} w_k(\mathbf{u}) p_{l+k}(\mathbf{u} + 2^{-k}\mathbf{z}). \tag{17}$$

Note that the stride option at neighboring layers is now represented by $2^{-k}$. For $k = -1$ with larger feature map size, the stride is 2 and for $k = 1$ with smaller feature map size, the stride is now $\frac{1}{2}$. If we apply a *jumping* action $S_m$ on the output PConv and leverage Eq. 15,

$$\begin{aligned} S_m(y^l(\mathbf{z})) &= S_m\left[\sum_{k=-1}^{1} [w_k * [p_{l+k}]](2^{-k}\mathbf{z})\right] \\ &= \sum_{k=-1}^{1} [w_k * [p_{l+k+m}]](2^{m-k}\mathbf{z}) \\ &= \sum_{k=-1}^{1} w_k * [S_m[p_{l+k}]](2^{-k}\mathbf{z}) \end{aligned} \tag{18}$$

The above equation shows an important property of using PConv in a Gaussian pyramid. That is, PConv commutes with the *jumping* action on the pyramid, which is conventionally called scale equivariance. It can be rephrased in another way. When the scale of an object changes in the original image, the extracted feature can also be found by shifting the convolved pyramid after using PConv, which also fits into the usual definition of scale invariance in object detection[2].

## 4. Experiment details

### 4.1. Training details

We trained the model with backbone ResNet-50 and ResNet-101 and mini-batch size of 16 on 8 Nvidia Titan XP GPUs. The training budget for the strategy `1x` was 12 epochs. The initial learning rate was 0.01, and was decreased by 0.1 after 8 and 11 epochs. When the `2x` schedule was adapted, we used 24 epochs for training and kept the same learning rate and decreased it by 0.1 after 16 and 22 epochs. All models with ResNext101-64-4d backbone were trained on Nvidia V100 GPUs under the same setting. When using BN in all experiments,we set 4 images per gpu with the same batch size to get more accurate statistics.

In the experiments of evaluating other feature fusion modules, all models used the same backbone ResNet-50 with 1x schedule. We used 4 PConvs in a combined way with one extra head to get better trade-off.In the HRNet,PANet, and Libra, and only replaced the origin FPN with the feature fusion module in the origin paper. In NAS-FPN,we used 7 merging-cells and keep channel 256.

### 4.2. Speed test details

We compared the speed of our method (include pre-precossing, forwarding and nms) with other proposed one stage detectors. All evaluation was performed on one Nvidia 1080Ti GPU with i7-7700k@4.2GHz. We set batch size to 8 and started the timer at 100-th iteration to make sure I/O is stable. Then we used the means of next 200 iteration in computation of speed.

## 5. Supplementary ablation experiments

### 5.1. Effect of the number of pyramid convolution stacks

The total number of pyramid convolutions is adjusted from 2 to 6. The recorded AP of different detectors is shown in Fig. 2. The figure illustrates that all these three detectors benefit from increasing the number of PConv from 2 to 4 due to the gradual information flow from top to bottom by PConv stacks. Using four stacked PConv in heads is rational as it provides descent average precision without causing much redundancy.

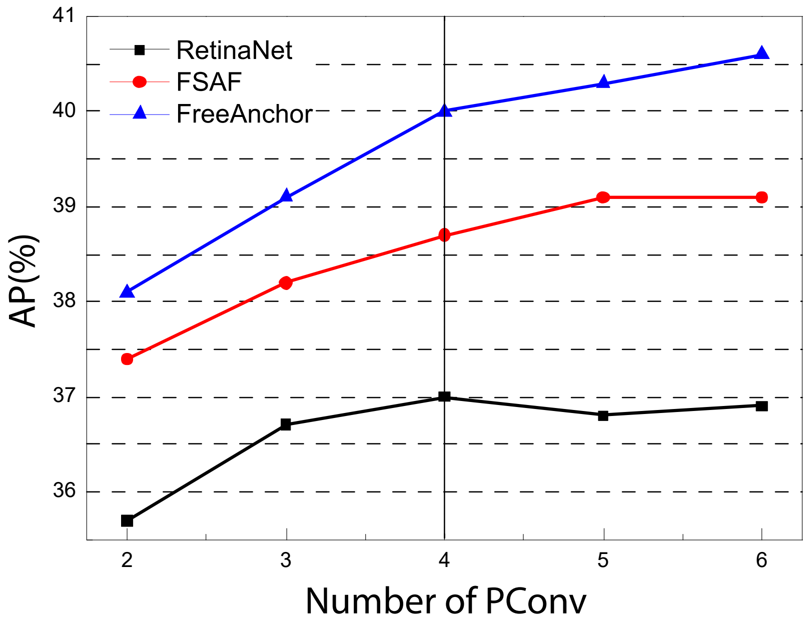

Figure 2: AP change with number of pyramid convolution in different architectures.

### 5.2. Training curves using iBN

Fig. 3(a-c) displays the training loss of different models and Fig. 3(d-f) presents how AP changes as training goes.

When iBN is used, in general the losses reduce faster in the early stage of training especially for FreeaAchor and RetinaNet. However, at the end of training, models with iBN result in a slightly higher training loss, yet the mAP is higher than models without IBN. This observation follows the better generalization property of batch normalization.

## 6. Details of FSAF

We only implemented the anchor-free branch in original paper. In the re-implementation process, we found that in label assigning phase, removing the ignore region could effectively improve model's ability to distinguish between positive and hard negative samples. Thus, in experiments, we set effective area as regions within 0.2 from center of projected area of object on the feature map, and assigned negative labels to all outside areas. This effort resulted in 1.1mAP improvement.

## References

[1] 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. 1

[2] 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. 4

[3] Tony Lindeberg. Scale-space theory: A basic tool for analyzing structures at different scales. *Journal of applied statistics*, 21(1-2):225–270, 1994. 3

[4] Daniel E Worrall and Max Welling. Deep scale-spaces: Equivariance over scale. In *Neural Information Processing Systems*, 2019. 3

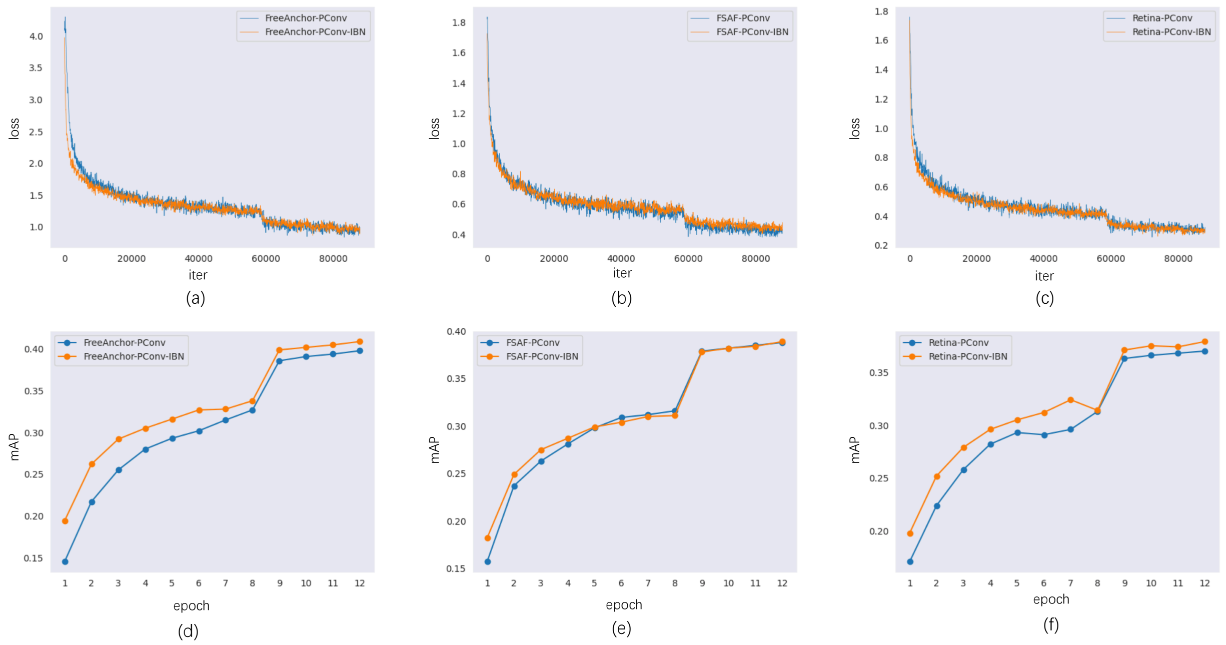

Figure 3: Comparison of training loss and validation AP with and without iBN. (a-c) shows the training loss of FreeAnchor, FSAF and RetinaNet, and (d-f) are the validation AP of FreeAnchor, FSAF and RetinaNet, respectively.