Rethinking Features-Fused-Pyramid-Neck for Object Detection Supplementary Material

Hulin Li

College of Traffic and Transportation, Chongqing Jiaotong University, Chongqing 400074, China
alan@mails.cqjtu.edu.cn

1 Experimental details

The SYolo is trained on eight RTX3090 GPUs and implementation by PyTorch (version 1.12.0). The training hyperparameters are in Tab. 1. We specifically recommend using the SGD optimizer for training SYolo, as SGD has consistently demonstrated significant advantages in numerous experiments, including faster convergence and higher performance ceilings. Complete training details will be released at https://github.com/AlanLi1997/rethinking-fpn, where you can use our provided default hyperparameters to train and reproduce results directly.

Table 1: Training hyperparameters

hyp.	value
learning rate (lr)	0.01→0.0001
lr scheduler	step decay
optimizer	SGD
momentum	0.937
weight decay	0.0005
batch size	16 per GPU
epochs	450
warmup epochs	3
warmup momentum	0.8
warmup bias lr	0.1

2 Core source codes

The core source codes for reproducing the results of this work are presented in this section. The complete project codes will be released at https://github.com/AlanLi1997/rethinking-fpn.

2.1 Code of the SNI

```
import torch
import torch.nn as nn

class SNI(nn.Module):
    # soft nearest neighbor interpolation for up-sampling
    # secondary features aligned

    def __init__(self, up_f=2):
        super().__init__()
        self.alpha = 1/(up_f**2)
        self.us = nn.Upsample(None, up_f, 'nearest')

    def forward(self, x):

        return self.alpha*self.us(x)
```

2.2 Code of the ESD

Code of the ESD-I

```
class Conv(nn.Module):
    # conv_bn_act

    def __init__(self, c1, c2, k=1, s=1, p=None, g=1, d=1, act=True):
        super().__init__()
        self.conv = nn.Conv2d(c1, c2, k, s, p, groups=g, dilation=d, bias
            =False)
        self.bn = nn.BatchNorm2d(c2)
        self.act = nn.SiLU() if act else nn.Identity()

    def forward(self, x):

        return self.act(self.bn(self.conv(x)))
```

```
class ESD1(nn.Module):
    # Extended spatial window for down-sampling
    # lightweight fusion

    def __init__(self, c1, c2, k=3, s=2, g=1, d=1, act=True):
        super().__init__()
        self.out_c = c2
        self.dense_feature = Conv(c1, c2, k, s, k // 2, g, d, act)  #
            window_dense_f
        self.global_feature = nn.AvgPool2d(4, 2, 1)  # window_global_f
        self.local_feature = nn.MaxPool2d(4, 2, 1)  # window_local_f

    def forward(self, x):
```

```
13
14          if self.out_c == x.shape[1]:
15              return self.global_feature(x) + self.local_feature(x) + self.
                    dense_feature(x)
16          else:
17              return torch.cat((self.global_feature(x), self.local_feature(
                    x)), dim=1) + self.dense_feature(x)
```

Code of the ESD-II

```
1  class ESD2(nn.Module):
2      # Extended spatial window for down-sampling
3      # learnable linearly fusion
4
5      def __init__(self, c1, c2, k=3, s=2, g=1, d=1, act=True):
6          super().__init__()
7          self.dense_feature = Conv(c1, c2, k, s, None, g, d, act)  #
               window_dense_f
8          self.global_feature = nn.AvgPool2d(4, 2, 1)  # window_global_f
9          self.local_feature = nn.MaxPool2d(4, 2, 1)  # window_local_f
10         self.fuse = nn.Conv2d(2*c2, c2, 1, 1, bias=False)
11
12     def forward(self, x):
13
14         return self.fuse(torch.cat((self.global_feature(x), self.
               local_feature(x), self.dense_feature(x)), dim=1))
```

2.3 Code of the GSConvE

Code of the GSConvE-I

```
1  class GSConvE1(nn.Module):
2      # enhancement lightweight conv
3
4      def __init__(self, c1, c2, k=1, s=1, g=1, d=1, act=True):
5          super().__init__()
6          c_ = c2 // 2
7          self.cv1 = Conv(c1, c_, k, s, k // 2, g, d, act)
8          self.cv2 = nn.Sequential(
9              nn.Conv2d(c_, c_, 3, 1, 1, bias=False),
10             nn.Conv2d(c_, c_, 3, 1, 1, groups=c_, bias=False),
11             nn.GELU()
12         )
13
14     def forward(self, x):
15         y = torch.cat((self.cv1(x), self.cv2(self.cv1(x))), dim=1)
16         # shuffle
```

```
        y = y.reshape(y.shape[0], 2, y.shape[1] // 2, y.shape[2], y.shape
            [3])
        output = y.permute(0, 2, 1, 3, 4)

        return output.reshape(output.shape[0], -1, output.shape[3],
            output.shape[4])
```

Code of the GSConvE-II

```
class GSConvE2(nn.Module):
    # enhancement lightweight conv

    def __init__(self, c1, c2, k=1, s=1, g=1, d=1, act=True):
        super().__init__()
        c_ = c2 // 4
        self.cv1 = Conv(c1, c_, k, s, k // 2, g, d, act)
        self.cv2 = Conv(c_, c_, 9, 1, k // 2, c_, d, act)
        self.cv3 = Conv(c_, c_, 13, 1, k // 2, c_, d, act)
        self.cv4 = Conv(c_, c_, 17, 1, k // 2, c_, d, act)

    def forward(self, x):
        y = torch.cat((self.cv1(x), self.cv2(self.cv1(x)), self.cv3(self.
            cv1(x)), self.cv4(self.cv1(x))), dim=1)
        # shuffle
        y = y.reshape(y.shape[0], 2, y.shape[1] // 2, y.shape[2], y.shape
            [3])
        output = y.permute(0, 2, 1, 3, 4)

        return output.reshape(output.shape[0], -1, output.shape[3],
            output.shape[4])
```