

PyramidBox 代码解析

GiantPandaCV 公众号出品

GiantPandaCV-zzk

2020-06-27



Contents

| | |
|---------------------------------|-----------|
| PyramidBox 代码解析 | 2 |
| 0. 序言 | 2 |
| 1. 简介 | 3 |
| 2. 骨干网络搭建 | 3 |
| 2.1 堆叠卷积块 | 4 |
| 2.2 Vgg 网络搭建 | 5 |
| 3. 低层特征金字塔网络 LFPN | 5 |
| 4. CPM 背景上下文感知模块 | 8 |
| 4.1 构造卷积 + 归一化层 | 11 |
| 4.2 CPM 模块搭建 | 12 |
| 5. L2 规范化 | 13 |
| 6. pyramidBox | 14 |
| 7. _vgg_ssd | 18 |
| 8. _vgg_ssd_loss | 20 |
| 9. train | 21 |
| 10. infer | 22 |
| 11. Data-anchor-sampling | 22 |
| 12. 总结 | 29 |

PyramidBox 代码解析

0. 序言

版权声明：此份电子书整理自公众号「GiantPandaCV」, 版权所有 GiantPandaCV, 禁止任何形式的转载, 禁止传播、商用, 违者必究! GiantPandaCV 公众号由专注于技术的一群 90 后创建, 专注于机器学习、深度学习、计算机视觉、图像处理等领域。半年以来已更新 **242** 篇原创技术文章。我们编写了

《从零开始学习 YOLOv3》、《从零开始学习 SSD》、《Faster R-CNN 原理和代码讲解》、《多目标跟踪快速入门》等系列原创电子书，关注后回复对应关键字即可免费领取。每天更新一到两篇相关推文，希望在传播知识、分享知识的同时能够启发你。欢迎扫描下方二维码关注我们的公众号。



1. 简介

本次要解析的是百度的一款人脸检测器，PyramidBox，官方代码开源在 Paddle 模型库里，https://github.com/PaddlePaddle/models/tree/release/1.8/PaddleCV/face_detection

由于该论文是 18 年提出的，所以代码形式都是以静态图格式来编写的

2. 骨干网络搭建

骨干网络主要是对 VGG16 进行一定的改造，作者保留了原始 VGG16 中的 conv1_1 到 pool5 这些层，把后续的两个全连接层，都转换为卷积 + 全连接层，其实也就是卷积层 + 1x1 卷积

下面我们来看下具体代码实现 (pyramidbox.py)

2.1 堆叠卷积块

熟悉 VGG16 网络的都应该知道该网络提出以一个模块进行堆叠获得更好的性能，在代码里同样也实现了一个方法来对卷积块进行堆叠

```
def conv_block(input, groups, filters, ksizes, strides=None, with_pool=True):
    assert len(filters) == groups
    assert len(ksizes) == groups
    strides = [1] * groups if strides is None else strides
    w_attr = ParamAttr(learning_rate=1., initializer=Xavier())
    b_attr = ParamAttr(learning_rate=2., regularizer=L2Decay(0.))
    conv = input
    for i in six.moves.xrange(groups):
        conv = fluid.layers.conv2d(
            input=conv,
            num_filters=filters[i],
            filter_size=ksizes[i],
            stride=strides[i],
            padding=(ksizes[i] - 1) // 2,
            param_attr=w_attr,
            bias_attr=b_attr,
            act='relu')
    if with_pool:
        pool = fluid.layers.pool2d(
            input=conv,
            pool_size=2,
            pool_type='max',
            pool_stride=2,
            ceil_mode=True)
        return conv, pool
    else:
        return conv
```

conv_block 参数解释如下

- input 输入张量
- groups 卷积层堆叠次数
- filters 卷积核个数，列表形式
- ksizes 卷积核大小，列表形式

- strides 步长
- with_pool 是否使用池化层

首先用 `assert` 语句保证卷积核个数和卷积核大小列表数量与卷积层堆叠次数一致

进入 `for` 循环，调用 `conv2d` 函数进行卷积操作，最后根据参数 `with_pool` 来决定是否使用最大池化操作，由于后面的 `fpn` 网络需要使用到卷积层，因此一并把卷积后的结果返回

2.2 Vgg 网络搭建

相关代码在 `PyramidBox` 类里面的 `_vgg` 函数里

```
def _vgg(self):
    self.conv1, self.pool1 = conv_block(self.image, 2, [64] * 2, [3] * 2)
    self.conv2, self.pool2 = conv_block(self.pool1, 2, [128] * 2, [3] * 2)

    #priorbox min_size is 16
    self.conv3, self.pool3 = conv_block(self.pool2, 3, [256] * 3, [3] * 3)
    #priorbox min_size is 32
    self.conv4, self.pool4 = conv_block(self.pool3, 3, [512] * 3, [3] * 3)
    #priorbox min_size is 64
    self.conv5, self.pool5 = conv_block(self.pool4, 3, [512] * 3, [3] * 3)

    # fc6 and fc7 in paper, priorbox min_size is 128
    self.conv6 = conv_block(
        self.pool5, 2, [1024, 1024], [3, 1], with_pool=False)
    # conv6_1 and conv6_2 in paper, priorbox min_size is 256
    self.conv7 = conv_block(
        self.conv6, 2, [256, 512], [1, 3], [1, 2], with_pool=False)
    # conv7_1 and conv7_2 in paper, priorbox mini_size is 512
    self.conv8 = conv_block(
        self.conv7, 2, [128, 256], [1, 3], [1, 2], with_pool=False)
```

这里就是调用前面的 `conv_block` 函数执行卷积操作

3. 低层特征金字塔网络 LFPN

网络示意图如下

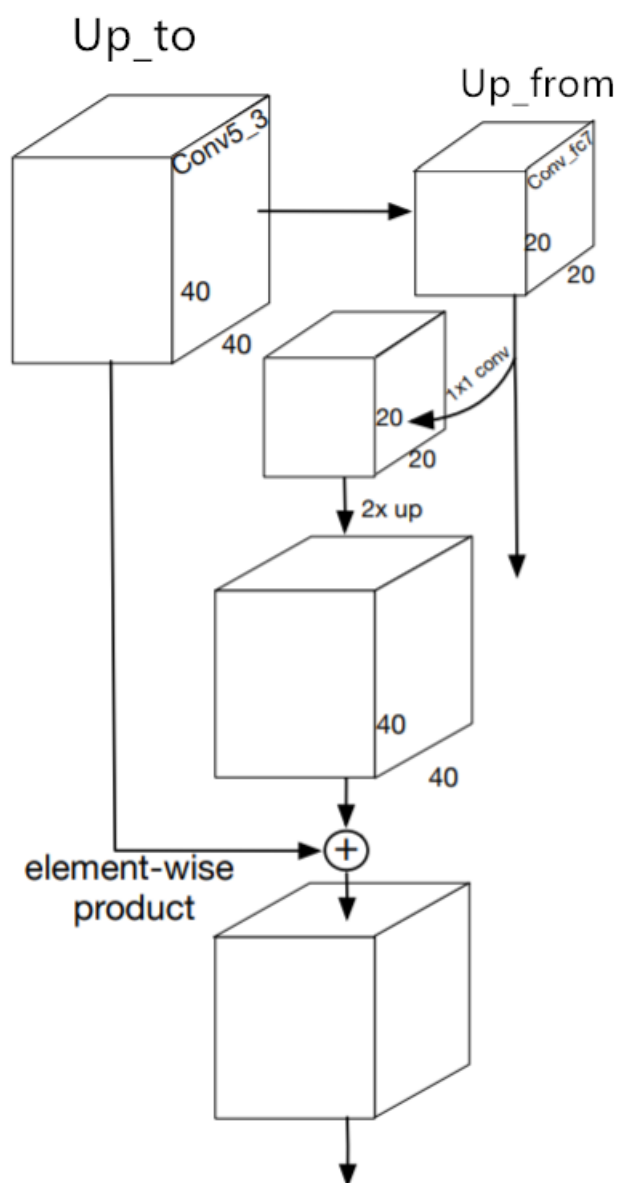


Figure 1: image-20200621115125857

该模块主要是以下三个操作

- 高层的特征图使用 1x1 卷积调整通道数
- 对高层的特征图进行上采样
- 与低层的特征图进行融合

相关代码在 PyramidBox 类的 `_low_level_fpn` 函数里

```
def _low_level_fpn(self):
    """
    Low-level feature pyramid network.
    """

    def fpn(up_from, up_to):
        ch = up_to.shape[1]
        b_attr = ParamAttr(learning_rate=2., regularizer=L2Decay(0.))
        conv1 = fluid.layers.conv2d(
            up_from, ch, 1, act='relu', bias_attr=b_attr)
        if self.use_transposed_conv2d:
            w_attr = ParamAttr(
                learning_rate=0.,
                regularizer=L2Decay(0.),
                initializer=Bilinear())
            upsampling = fluid.layers.conv2d_transpose(
                conv1,
                ch,
                output_size=None,
                filter_size=4,
                padding=1,
                stride=2,
                groups=ch,
                param_attr=w_attr,
                bias_attr=False,
                use_cudnn=False)
        else:
            upsampling = fluid.layers.resize_bilinear(
                conv1, out_shape=up_to.shape[2:])

        conv2 = fluid.layers.conv2d(
            up_to, ch, 1, act='relu', bias_attr=b_attr)
        if self.is_infer:
            upsampling = fluid.layers.crop(upsampling, shape=conv2)
        # eltwise mul
        conv_fuse = upsampling * conv2
        return conv_fuse

    self.lfpn2_on_conv5 = fpn(self.conv6, self.conv5)
    self.lfpn1_on_conv4 = fpn(self.lfpn2_on_conv5, self.conv4)
    self.lfpn0_on_conv3 = fpn(self.lfpn1_on_conv4, self.conv3)
```

首先里面构造了一个方法 `fpn`，参数解释如下

- `up_from` 高层级的特征图（即分辨率更低的那个特征图）
- `up_to` 低层级的特征图（即分辨率更高的那个特征图）

`ch` 获得低层级特征图的通道数目

上采样的方式有以下两种

1. 使用反卷积模块进行上采样
2. 直接使用双线性插值进行上采样

这里是根据类属性 `self.use_transposed_conv2d` 进行决定

另外代码实现也与图示有一定区别，这里代码又对 `up_to` 这个特征图做了个 `1x1` 的卷积，得到 `conv2` 属性 `is_infer`，如果是在推理阶段，还对上采样特征图进行一次裁剪

最后将 `conv2` 和 `upsampling` 这两个特征图进行 `elementwise_mul` 相乘

到这里我们就实现了 FPN 操作

而 LFPN 并不是从最后一层特征图进行特征融合，而是从中间选取了三组进行特征融合

这里只对 `conv5`, `conv4`, `conv3` 进行 FPN 操作

4. CPM 背景上下文感知模块

在每一个 `conv` 模块后，都会连接 LFPN 模块和 CPM 模块（没有 LFPN 的卷积层直接连 CPM 模块）

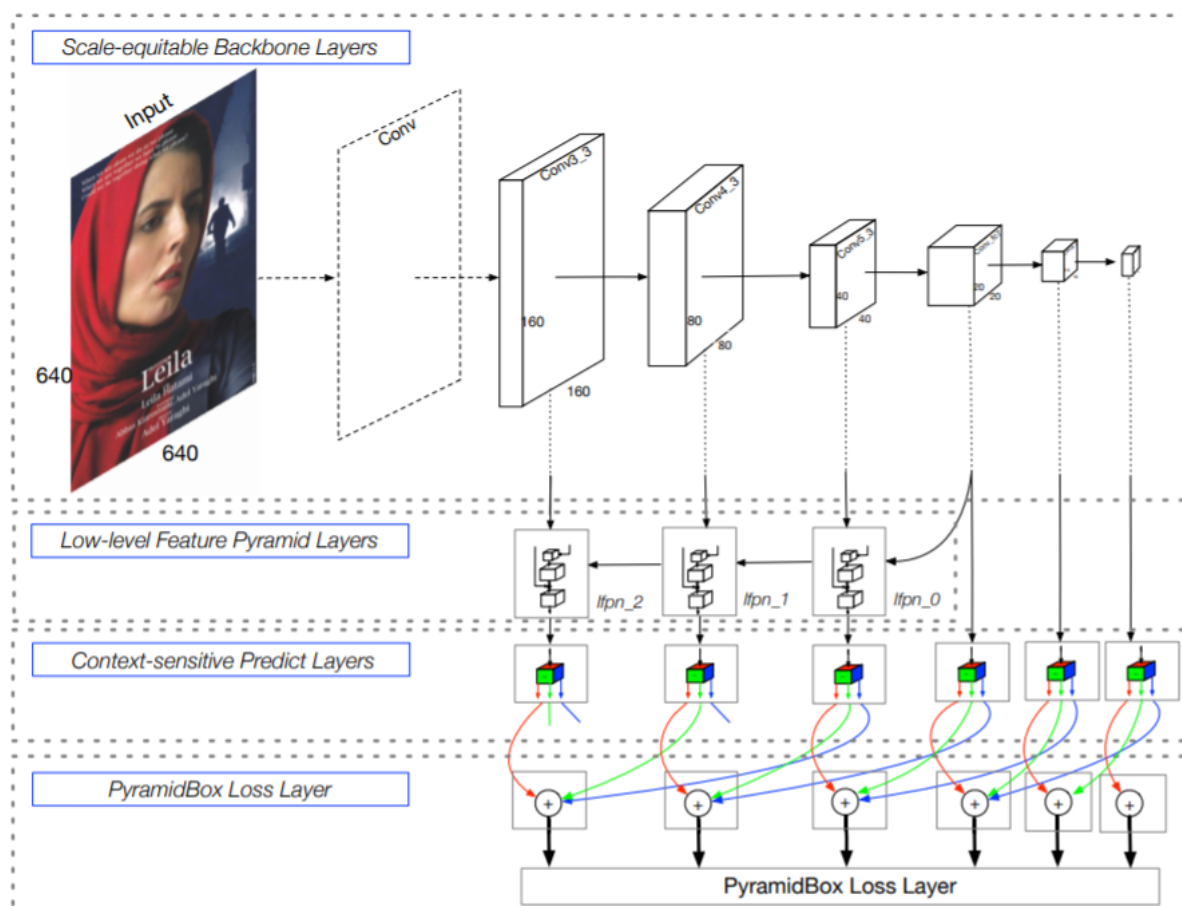


Figure 2: image-20200621123545167

从形式上看，CPM 模块一共有 3 个分支，分别与当前和前面两层合并在一起计算损失

CPM 模块结构构造如下

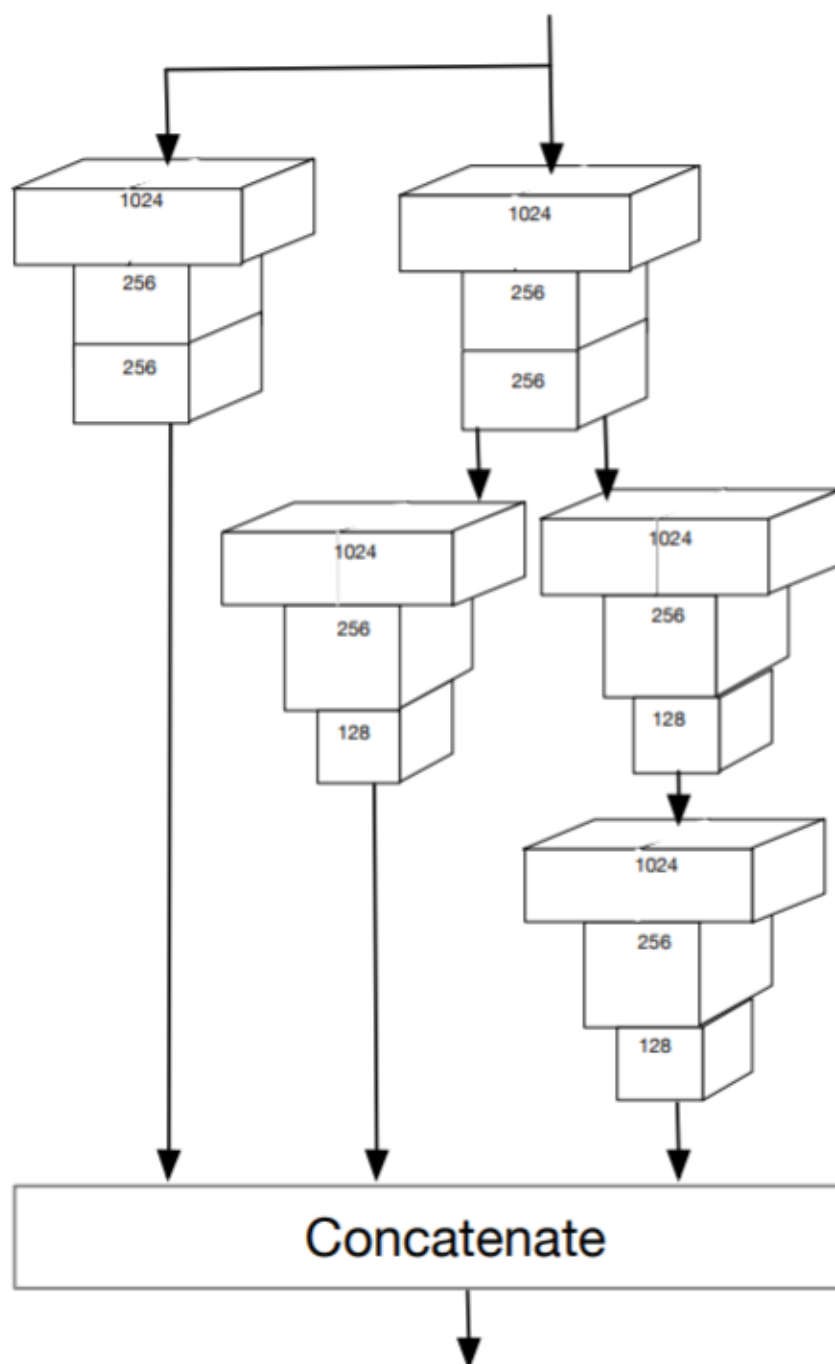


Figure 3: image-20200621123503037

论文将其中的模块替代成 **SSH** 和 **DSSD** 这两个模块的组合，SSH 模块通过堆叠卷积核让计算的特征

具有不同的感受野，DSSD 则引入残差结构。两种结构的结合让整个模块更深更宽。

4.1 构造卷积 + 归一化层

CPM 模块这里的卷积是带有归一化层的，这里我们先实现卷积归一化层

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import numpy as np
import six
import paddle.fluid as fluid
from paddle.fluid.param_attr import ParamAttr
from paddle.fluid.initializer import Xavier
from paddle.fluid.initializer import Constant
from paddle.fluid.initializer import Bilinear
from paddle.fluid.regularizer import L2Decay

def conv_bn(input, filter, ksize, stride, padding, act='relu',
            ↪ bias_attr=False):
    p_attr = ParamAttr(learning_rate=1., regularizer=L2Decay(0.))
    b_attr = ParamAttr(learning_rate=0., regularizer=L2Decay(0.))
    conv = fluid.layers.conv2d(
        input=input,
        filter_size=ksize,
        num_filters=filter,
        stride=stride,
        padding=padding,
        act=None,
        bias_attr=bias_attr)
    return fluid.layers.batch_norm(
        input=conv,
        act=act,
        epsilon=0.001,
        momentum=0.999,
        param_attr=p_attr,
        bias_attr=b_attr)
```

先导入相关的包，conv_bn 这个函数的参数解释如下

- input 输入张量
- filter 卷积核个数
- ksize 卷积核大小
- stride 步长大小
- padding 填充大小
- act 激活方式，默认是 relu 激活
- bias_attr 是否给卷积层添加偏置

首先定义 p_attr, b_attr 这两个参数，用于归一化层的初始化

然后调用 conv2d 来执行卷积操作，最后返回归一化，激活后的输出张量

4.2 CPM 模块搭建

```
def _cpm_module(self):
    """
    Context-sensitive Prediction Module
    """

    def cpm(input):
        # residual
        branch1 = conv_bn(input, 1024, 1, 1, 0, None)
        branch2a = conv_bn(input, 256, 1, 1, 0, act='relu')
        branch2b = conv_bn(branch2a, 256, 3, 1, 1, act='relu')
        branch2c = conv_bn(branch2b, 1024, 1, 1, 0, None)
        sum = branch1 + branch2c
        rescomb = fluid.layers.relu(x=sum)

        # ssh
        b_attr = ParamAttr(learning_rate=2., regularizer=L2Decay(0.))
        ssh_1 = fluid.layers.conv2d(rescomb, 256, 3, 1, 1,
        ↪ bias_attr=b_attr)
        ssh_dimred = fluid.layers.conv2d(
            rescomb, 128, 3, 1, 1, act='relu', bias_attr=b_attr)
        ssh_2 = fluid.layers.conv2d(
            ssh_dimred, 128, 3, 1, 1, bias_attr=b_attr)
        ssh_3a = fluid.layers.conv2d(
            ssh_dimred, 128, 3, 1, 1, act='relu', bias_attr=b_attr)
        ssh_3b = fluid.layers.conv2d(ssh_3a, 128, 3, 1, 1,
        ↪ bias_attr=b_attr)

        ssh_concat = fluid.layers.concat([ssh_1, ssh_2, ssh_3b], axis=1)
```

```

        ssh_out = fluid.layers.relu(x=ssh_concat)
        return ssh_out

    self.ssh_conv3 = cpm(self.lfpn0_on_conv3)
    self.ssh_conv4 = cpm(self.lfpn1_on_conv4)
    self.ssh_conv5 = cpm(self.lfpn2_on_conv5)
    self.ssh_conv6 = cpm(self.conv6)
    self.ssh_conv7 = cpm(self.conv7)
    self.ssh_conv8 = cpm(self.conv8)

```

替换后的结构图应该如下

上半部分是 DSSD 残差结构

下半部分则是 SSH 结构

它这里使用的是 DSSD 和 SSH 两个模块的结合，DSSD 通过设计残差结构加强模型表达，SSH 通过对分支进行不同程度的卷积块堆叠，进行不同感受野的运算。

5. L2 规范化

这是在 SSD 中做的一个操作，为了让拼接后的向量量纲一致，需要对其做一个 L2 规范化的操作，代码如下

```

def _l2_norm_scale(self, input, init_scale=1.0, channel_shared=False):
    from paddle.fluid.layer_helper import LayerHelper
    helper = LayerHelper("Scale")
    l2_norm = fluid.layers.l2_normalize(
        input, axis=1) # l2 norm along channel
    shape = [1] if channel_shared else [input.shape[1]]
    scale = helper.create_parameter(
        attr=helper.param_attr,
        shape=shape,
        dtype=input.dtype,
        default_initializer=Constant(init_scale))
    out = fluid.layers.elementwise_mul(
        x=l2_norm, y=scale, axis=-1 if channel_shared else 1)
    return out

```

这里先调用 l2_normalize 进行 L2 规范化

然后为变量 scale 设置一个可训练参数，作为缩放参数，初始化为 1.0

将 L2 规范化后的值与 scale 进行 elementwise_mul 相乘，进行缩放

6. pyramidBox

接下来到了最精华的部分 pyramidBox

```
def _pyramidbox(self):
    """
    Get prior-boxes and pyramid-box
    """
    self.ssh_conv3_norm = self._l2_norm_scale(
        self.ssh_conv3, init_scale=10.)
    self.ssh_conv4_norm = self._l2_norm_scale(self.ssh_conv4,
        ↪ init_scale=8.)
    self.ssh_conv5_norm = self._l2_norm_scale(self.ssh_conv5,
        ↪ init_scale=5.)

    def permute_and_reshape(input, last_dim):
        trans = fluid.layers.transpose(input, perm=[0, 2, 3, 1])
        compile_shape = [
            trans.shape[0], np.prod(trans.shape[1:]) // last_dim, last_dim
        ]
        run_shape = fluid.layers.assign(
            np.array([0, -1, last_dim]).astype("int32"))
        return fluid.layers.reshape(
            trans, shape=compile_shape, actual_shape=run_shape)

    face_locs, face_confs = [], []
    head_locs, head_confs = [], []
    boxes, vars = [], []

    b_attr = ParamAttr(learning_rate=2., regularizer=L2Decay(0.))
    mbox_loc = fluid.layers.conv2d(
        self.ssh_conv3_norm, 8, 3, 1, 1, bias_attr=b_attr)
    face_loc, head_loc = fluid.layers.split(
        mbox_loc, num_or_sections=2, dim=1)
    face_loc = permute_and_reshape(face_loc, 4)
    if not self.is_infer:
        head_loc = permute_and_reshape(head_loc, 4)

    mbox_conf = fluid.layers.conv2d(
        self.ssh_conv3_norm, 8, 3, 1, 1, bias_attr=b_attr)
    face_conf3, face_conf1, head_conf3, head_conf1 = fluid.layers.split(
        mbox_conf, num_or_sections=[3, 1, 3, 1], dim=1)
    face_conf3_maxin = fluid.layers.reduce_max(
```

```
        face_conf3, dim=1, keep_dim=True)
    face_conf = fluid.layers.concat([face_conf3_maxin, face_conf1],
↪   axis=1)
    face_conf = permute_and_reshape(face_conf, 2)
    if not self.is_infer:
        head_conf3_maxin = fluid.layers.reduce_max(
            head_conf3, dim=1, keep_dim=True)
        head_conf = fluid.layers.concat(
            [head_conf3_maxin, head_conf1], axis=1)
        head_conf = permute_and_reshape(head_conf, 2)

    face_locs.append(face_loc)
    face_confs.append(face_conf)
    if not self.is_infer:
        head_locs.append(head_loc)
        head_confs.append(head_conf)

    box, var = fluid.layers.prior_box(
        self.ssh_conv3_norm,
        self.image,
        min_sizes=[16.],
        steps=[4.] * 2,
        aspect_ratios=[1.],
        clip=False,
        flip=True,
        offset=0.5)
    box = fluid.layers.reshape(box, shape=[-1, 4])
    var = fluid.layers.reshape(var, shape=[-1, 4])
    boxes.append(box)
    vars.append(var)

    inputs = [
        self.ssh_conv4_norm, self.ssh_conv5_norm, self.ssh_conv6,
        self.ssh_conv7, self.ssh_conv8
    ]
    for i, input in enumerate(inputs):
        mbox_loc = fluid.layers.conv2d(input, 8, 3, 1, 1,
↪   bias_attr=b_attr)
        face_loc, head_loc = fluid.layers.split(
            mbox_loc, num_or_sections=2, dim=1)
        face_loc = permute_and_reshape(face_loc, 4)
        if not self.is_infer:
            head_loc = permute_and_reshape(head_loc, 4)
```

```
mbox_conf = fluid.layers.conv2d(input, 6, 3, 1, 1,
↪ bias_attr=b_attr)
    face_conf1, face_conf3, head_conf = fluid.layers.split(
        mbox_conf, num_or_sections=[1, 3, 2], dim=1)
    face_conf3_maxin = fluid.layers.reduce_max(
        face_conf3, dim=1, keep_dim=True)
    face_conf = fluid.layers.concat(
        [face_conf1, face_conf3_maxin], axis=1)

    face_conf = permute_and_reshape(face_conf, 2)
    if not self.is_infer:
        head_conf = permute_and_reshape(head_conf, 2)

    face_locs.append(face_loc)
    face_confs.append(face_conf)

    if not self.is_infer:
        head_locs.append(head_loc)
        head_confs.append(head_conf)

    box, var = fluid.layers.prior_box(
        input,
        self.image,
        min_sizes=[self.min_sizes[i + 1]],
        steps=[self.steps[i + 1]] * 2,
        aspect_ratios=[1.],
        clip=False,
        flip=True,
        offset=0.5)
    box = fluid.layers.reshape(box, shape=[-1, 4])
    var = fluid.layers.reshape(var, shape=[-1, 4])

    boxes.append(box)
    vars.append(var)

    self.face_mbox_loc = fluid.layers.concat(face_locs, axis=1)
    self.face_mbox_conf = fluid.layers.concat(face_confs, axis=1)

    if not self.is_infer:
        self.head_mbox_loc = fluid.layers.concat(head_locs, axis=1)
        self.head_mbox_conf = fluid.layers.concat(head_confs, axis=1)
```



```
self.prior_boxes = fluid.layers.concat(boxes)
self.box_vars = fluid.layers.concat(vars)
```

作者先对第 3, 4, 5 的 ssh 层进行 L2 规范化

perute_and_reshape 方法是与 SSD 算法一致的, 为了连结多尺度的预测结果, 先将通道维放在最后一维上, 后面 reshape 成二维形式 (批量大小, 高 \times 宽 \times 通道维)

face_locs, face_confs, head_locs, head_confs 分别代表脸部位置, 脸部置信度, 头部位置, 头部置信度

mbox_loc 先由 ssh_conv3_norm 作为输入, 以 8 个通道维输出, 并 split 成 4, 4 的形式给 face_loc, head_loc, 以此表示面部, 头部的坐标

论文中为了解决 Unbalance 的问题, 引入 **Max-in-Max-out**

灵感是来自 **S3FD** 算法中的 **Max-out**, 当锚框很小的时候, 会有很多负类, 为了平衡, 就不单纯做二分类, 可以把负类分成负 1, 负 2, 负 3 等多类, 再从负类中选取置信度最高的那个拿出来, 一定程度上缓解了样本不均衡问题。

而 Max-in-Max-out 不仅对负类做, 而且也对后续的卷积层 (预测的锚框更大) 中预测正类的也做了 Max-in

如下图公式

$$cp_l = \begin{cases} 1, & \text{if } l = 0, \\ 3, & \text{otherwise.} \end{cases}$$

Figure 4: image-20200627101159046

参数 L 代表的是预测锚框的卷积层, $L=0$ 则是对第一个层, 取正类为 1, 换句话说就是对负类 max-out 其他层正类为 3, 也就是对正类 max-in

代码中先对第一个卷积层做处理

```
mbox_conf = fluid.layers.conv2d(
    self.ssh_conv3_norm, 8, 3, 1, 1, bias_attr=b_attr)
```

```

        face_conf3, face_conf1, head_conf3, head_conf1 = fluid.layers.split(
            mbox_conf, num_or_sections=[3, 1, 3, 1], dim=1)
        face_conf3_maxin = fluid.layers.reduce_max(
            face_conf3, dim=1, keep_dim=True)
        face_conf = fluid.layers.concat([face_conf3_maxin, face_conf1],
        ↪ axis=1)
        face_conf = permute_and_reshape(face_conf, 2)
        if not self.is_infer:
            head_conf3_maxin = fluid.layers.reduce_max(
                head_conf3, dim=1, keep_dim=True)
            head_conf = fluid.layers.concat(
                [head_conf3_maxin, head_conf1], axis=1)
            head_conf = permute_and_reshape(head_conf, 2)

        face_locs.append(face_loc)
        face_confs.append(face_conf)

```

先卷积得到 8 个通道，再分别以 3, 1, 3, 1 的形式分给 脸部负类置信度，脸部正类置信度，头部负类置信度，头部正类置信度。再调用 **reduce_max** 方法取的负类置信度最大值，然后与正类连结到一起，reshape。

最后通过 **prior_box** 内置的 OP（用于 SSD 算法），得到候选框信息以及对应的方差

后半部分代码也很简单，就是将后续的卷积层，通过 **for** 循环统一进行处理，只不过与第一个卷积层相反，是对正类做 **maxin** 处理

7. _vgg_ssd

这个方法就是 SSD 算法代码，没有将头部身体等预测加入，只是单独对脸部做检测，并同样使用了 max-in-out 策略。这与前面的 PyramidBox 类似，这里就不再展开讲解了

```

def _vgg_ssd(self):
    self.conv3_norm = self._l2_norm_scale(self.conv3, init_scale=10.)
    self.conv4_norm = self._l2_norm_scale(self.conv4, init_scale=8.)
    self.conv5_norm = self._l2_norm_scale(self.conv5, init_scale=5.)

    def permute_and_reshape(input, last_dim):
        trans = fluid.layers.transpose(input, perm=[0, 2, 3, 1])
        compile_shape = [
            trans.shape[0], np.prod(trans.shape[1:]) // last_dim, last_dim
        ]
        run_shape = fluid.layers.assign(

```

```

        np.array([0, -1, last_dim]).astype("int32"))
    return fluid.layers.reshape(
        trans, shape=compile_shape, actual_shape=run_shape)

locs, confs = [], []
boxes, vars = [], []
b_attr = ParamAttr(learning_rate=2., regularizer=L2Decay(0.))

# conv3
mbox_loc = fluid.layers.conv2d(
    self.conv3_norm, 4, 3, 1, 1, bias_attr=b_attr)
loc = permute_and_reshape(mbox_loc, 4)
mbox_conf = fluid.layers.conv2d(
    self.conv3_norm, 4, 3, 1, 1, bias_attr=b_attr)
conf1, conf3 = fluid.layers.split(
    mbox_conf, num_or_sections=[1, 3], dim=1)
conf3_maxin = fluid.layers.reduce_max(conf3, dim=1, keep_dim=True)
conf = fluid.layers.concat([conf1, conf3_maxin], axis=1)
conf = permute_and_reshape(conf, 2)
box, var = fluid.layers.prior_box(
    self.conv3_norm,
    self.image,
    min_sizes=[16.],
    steps=[4, 4],
    aspect_ratios=[1.],
    clip=False,
    flip=True,
    offset=0.5)
box = fluid.layers.reshape(box, shape=[-1, 4])
var = fluid.layers.reshape(var, shape=[-1, 4])

locs.append(loc)
confs.append(conf)
boxes.append(box)
vars.append(var)

min_sizes = [32., 64., 128., 256., 512.]
steps = [8., 16., 32., 64., 128.]
inputs = [
    self.conv4_norm, self.conv5_norm, self.conv6, self.conv7,
    ↪ self.conv8
]
for i, input in enumerate(inputs):

```

```
mbox_loc = fluid.layers.conv2d(input, 4, 3, 1, 1,
↪ bias_attr=b_attr)
    loc = permute_and_reshape(mbox_loc, 4)

mbox_conf = fluid.layers.conv2d(input, 2, 3, 1, 1,
↪ bias_attr=b_attr)
    conf = permute_and_reshape(mbox_conf, 2)
    box, var = fluid.layers.prior_box(
        input,
        self.image,
        min_sizes=[min_sizes[i]],
        steps=[steps[i]] * 2,
        aspect_ratios=[1.],
        clip=False,
        flip=True,
        offset=0.5)
    box = fluid.layers.reshape(box, shape=[-1, 4])
    var = fluid.layers.reshape(var, shape=[-1, 4])

    locs.append(loc)
    confs.append(conf)
    boxes.append(box)
    vars.append(var)

self.face_mbox_loc = fluid.layers.concat(locs, axis=1)
self.face_mbox_conf = fluid.layers.concat(confs, axis=1)
self.prior_boxes = fluid.layers.concat(boxes)
self.box_vars = fluid.layers.concat(vars)
```

8. _vgg_ssd_loss

这个方法同样也是单独针对 ssd 算法的

通过调用 paddle 内置的 ssd_loss 计算损失

```
def vgg_ssd_loss(self):
    loss = fluid.layers.ssd_loss(
        self.face_mbox_loc,
        self.face_mbox_conf,
        self.face_box,
        self.gt_label,
```

```
        self.prior_boxes,
        self.box_vars,
        overlap_threshold=0.35,
        neg_overlap=0.35)
    loss = fluid.layers.reduce_sum(loss)
    loss.persistable = True
    return loss
```

9. train

这部分是训练主体函数，这里使用的是 pyramidbox 损失计算方法，分别计算脸部和头部的 loss，最后加在一起

```
def train(self):
    face_loss = fluid.layers.ssd_loss(
        self.face_mbox_loc,
        self.face_mbox_conf,
        self.face_box,
        self.gt_label,
        self.prior_boxes,
        self.box_vars,
        overlap_threshold=0.35,
        neg_overlap=0.35)
    face_loss.persistable = True
    head_loss = fluid.layers.ssd_loss(
        self.head_mbox_loc,
        self.head_mbox_conf,
        self.head_box,
        self.gt_label,
        self.prior_boxes,
        self.box_vars,
        overlap_threshold=0.35,
        neg_overlap=0.35)
    head_loss.persistable = True
    face_loss = fluid.layers.reduce_sum(face_loss)
    face_loss.persistable = True
    head_loss = fluid.layers.reduce_sum(head_loss)
    head_loss.persistable = True
    total_loss = face_loss + head_loss
    total_loss.persistable = True
    return face_loss, head_loss, total_loss
```

整体上也是采用 `ssd_loss` 计算方式，返回的分别是脸部 `loss`，头部 `loss`，以及 `loss` 总和

10. infer

这部分代码用于推理阶段，主要是输出 `nms` 后的值

```
def infer(self, main_program=None):
    if main_program is None:
        test_program = fluid.default_main_program().clone(for_test=True)
    else:
        test_program = main_program.clone(for_test=True)
    with fluid.program_guard(test_program):
        face_nmsed_out = fluid.layers.detection_output(
            self.face_mbox_loc,
            self.face_mbox_conf,
            self.prior_boxes,
            self.box_vars,
            nms_threshold=0.3,
            nms_top_k=5000,
            keep_top_k=750,
            score_threshold=0.01)
    return test_program, face_nmsed_out
```

这里的 `test_program` 指的是测试阶段的计算图，调用 `paddle` 内置的 `detection_output` 函数，将 `nms` 阈值设定为 0.35，将预测框输出

至此整体网络代码已经解读完毕，下面讲解的是 `pyramidBox` 引入的一个采样方法

11. Data-anchor-sampling

简单来说，这个采样方法是缩放了训练图片，通过图片中一个随机人脸 **reshape** 成一个随机更小的锚框大小。

整个网络锚框大小设定如下

$$s_i = 2^{4+i}, \text{ for } i = 0, 1, \dots, 5,$$

Figure 5: image-20200627121013525

也就是 16, 32, 64.....512

然后设 s_{face} 为人脸框大小 s_{anchor_i} 为第 i 个锚框大小

$$i_{anchor} = \operatorname{argmin}_i \operatorname{abs}(s_{anchor_i} - s_{face})$$

换句话说, i 是比人脸框稍大的那个锚框的索引 index

然后我们从

$$\{0, 1, \dots, \min(5, i_{anchor} + 1)\},$$

Figure 6: image-20200627121428165

随机选取一个数字 i_{target} , \min 中的 5 是因为我们锚框最大的 index 就是 5

再设置一个变量 s_{target}

$$s_{target} = \operatorname{random}(s_{i_{target}}/2, s_{i_{target}} * 2).$$

Figure 7: image-20200627122830213

随机从 $(s_{target}/2, s_{target}*2)$ 选取一个数

$$s^* = s_{target} / s_{face}.$$

Figure 8: image-20200627122926198

最后再与人脸区域相除，得到最后的缩放 `scale`

下面我举个例子

假设我的人脸大小是52x52

那么相邻两个锚框大小分别是32x32 和 64x64

经过比较52距离64更近一点，因此`ianchor = 2`（因为64x64这个锚框对应的`i`是2）

然后在`set`里面就是(0, 1, 2, 3)，随机选取1个数字`i_target`，假设这里选到的是1

`s_target = random(s1/2, s1*2) = random(16, 64)` 假设这里选到的是48

`s* = 48/52 = 0.923`

这就是最终的缩放大小

这部分的代码在 `image_util.py` 里

首先定义了一个 `sampler` 类，里面主要定义的是一些参数

```
class sampler():
    def __init__(self,
                  max_sample,
                  max_trial,
                  min_scale,
                  max_scale,
                  min_aspect_ratio,
                  max_aspect_ratio,
                  min_jaccard_overlap,
                  max_jaccard_overlap,
                  min_object_coverage,
                  max_object_coverage,
                  use_square=False):
        self.max_sample = max_sample
        self.max_trial = max_trial
        self.min_scale = min_scale
        self.max_scale = max_scale
        self.min_aspect_ratio = min_aspect_ratio
```



```
self.max_aspect_ratio = max_aspect_ratio
self.min_jaccard_overlap = min_jaccard_overlap
self.max_jaccard_overlap = max_jaccard_overlap
self.min_object_coverage = min_object_coverage
self.max_object_coverage = max_object_coverage
self.use_square = use_square
```

然后定义了一个类 `bbox`，表示锚框，即四个坐标

```
class bbox():
    def __init__(self, xmin, ymin, xmax, ymax):
        self.xmin = xmin
        self.ymin = ymin
        self.xmax = xmax
        self.ymax = ymax
```

接着是我们方法的主体

```
def data_anchor_sampling(sampler, bbox_labels, image_width, image_height,
                        scale_array, resize_width, resize_height):
    num_gt = len(bbox_labels)
    # np.random.randint range: [low, high)
    rand_idx = np.random.randint(0, num_gt) if num_gt != 0 else 0

    if num_gt != 0:
        norm_xmin = bbox_labels[rand_idx][1]
        norm_ymin = bbox_labels[rand_idx][2]
        norm_xmax = bbox_labels[rand_idx][3]
        norm_ymax = bbox_labels[rand_idx][4]

        xmin = norm_xmin * image_width
        ymin = norm_ymin * image_height
        wid = image_width * (norm_xmax - norm_xmin)
        hei = image_height * (norm_ymax - norm_ymin)
        range_size = 0

        area = wid * hei
        for scale_ind in range(0, len(scale_array) - 1):
            if area > scale_array[scale_ind] ** 2 and area < \
                scale_array[scale_ind + 1] ** 2:
                range_size = scale_ind + 1
                break

        if area > scale_array[len(scale_array) - 2]**2:
```

```

        range_size = len(scale_array) - 2

scale_choose = 0.0
if range_size == 0:
    rand_idx_size = 0
else:
    # np.random.randint range: [low, high)
    rng_rand_size = np.random.randint(0, range_size + 1)
    rand_idx_size = rng_rand_size % (range_size + 1)

if rand_idx_size == range_size:
    min_resize_val = scale_array[rand_idx_size] / 2.0
    max_resize_val = min(2.0 * scale_array[rand_idx_size],
                        2 * math.sqrt(wid * hei))
    scale_choose = random.uniform(min_resize_val, max_resize_val)
else:
    min_resize_val = scale_array[rand_idx_size] / 2.0
    max_resize_val = 2.0 * scale_array[rand_idx_size]
    scale_choose = random.uniform(min_resize_val, max_resize_val)

sample_bbox_size = wid * resize_width / scale_choose

w_off_orig = 0.0
h_off_orig = 0.0
if sample_bbox_size < max(image_height, image_width):
    if wid <= sample_bbox_size:
        w_off_orig = np.random.uniform(xmin + wid - sample_bbox_size,
                                        xmin)
    else:
        w_off_orig = np.random.uniform(xmin,
                                        xmin + wid - sample_bbox_size)

    if hei <= sample_bbox_size:
        h_off_orig = np.random.uniform(ymin + hei - sample_bbox_size,
                                        ymin)
    else:
        h_off_orig = np.random.uniform(ymin,
                                        ymin + hei - sample_bbox_size)
else:
    w_off_orig = np.random.uniform(image_width - sample_bbox_size,
    ↪ 0.0)

```

```

        h_off_orig = np.random.uniform(image_height - sample_bbox_size,
        ↪ 0.0)

        w_off_orig = math.floor(w_off_orig)
        h_off_orig = math.floor(h_off_orig)

        # Figure out top left coordinates.
        w_off = 0.0
        h_off = 0.0
        w_off = float(w_off_orig / image_width)
        h_off = float(h_off_orig / image_height)

        sampled_bbox = bbox(w_off, h_off,
                            w_off + float(sample_bbox_size / image_width),
                            h_off + float(sample_bbox_size / image_height))
        return sampled_bbox
    else:
        return 0

```

num_gt 先获得人脸框最大的索引

然后从中随机选取 1 个

由于 bbox 对应的标记都是归一化过的

所以我们取出来的 x,y 还需要乘上图片宽，高恢复

这里锚框采用的是 xywh 格式

所以通过 image_width*(xmax-xmin) 得到人脸框宽度，同理得到人脸框高度

area 就是我们的人脸大小

接下来的一段 for 循环，就是查找前面说过的 i_anchor 索引

```

    for scale_ind in range(0, len(scale_array) - 1):
        if area > scale_array[scale_ind] ** 2 and area < \
            scale_array[scale_ind + 1] ** 2:
            range_size = scale_ind + 1
            break

```

在最大值为 i_anchor 的索引范围内选取一个

```

        scale_choose = 0.0
        if range_size == 0:
            rand_idx_size = 0
        else:

```

```
# np.random.randint range: [low, high)
rng_rand_size = np.random.randint(0, range_size + 1)
rand_idx_size = rng_rand_size % (range_size + 1)
```

然后随机设置缩放大小，范围在 $\text{sitarget}/2$, $\text{sitarget} \times 2$

```
if rand_idx_size == range_size:
    min_resize_val = scale_array[rand_idx_size] / 2.0
    max_resize_val = min(2.0 * scale_array[rand_idx_size],
                          2 * math.sqrt(wid * hei))
    scale_choose = random.uniform(min_resize_val, max_resize_val)
else:
    min_resize_val = scale_array[rand_idx_size] / 2.0
    max_resize_val = 2.0 * scale_array[rand_idx_size]
    scale_choose = random.uniform(min_resize_val, max_resize_val)
```

这里还做了个特殊处理，当 `randsize` 刚好等于最大的值，需要保证其最大缩放不超过原始人脸的两倍

`scale_choose` 就是在 `minresize` 和 `maxresize` 中以正态分布形式随机选取

再做一个随机偏移

```
if sample_bbox_size < max(image_height, image_width):
    if wid <= sample_bbox_size:
        w_off_orig = np.random.uniform(xmin + wid - sample_bbox_size,
                                         xmin)
    else:
        w_off_orig = np.random.uniform(xmin,
                                         xmin + wid - sample_bbox_size)

    if hei <= sample_bbox_size:
        h_off_orig = np.random.uniform(ymin + hei - sample_bbox_size,
                                         ymin)
    else:
        h_off_orig = np.random.uniform(ymin,
                                         ymin + hei - sample_bbox_size)

else:
    w_off_orig = np.random.uniform(image_width - sample_bbox_size,
    ↪ 0.0)
    h_off_orig = np.random.uniform(image_height - sample_bbox_size,
    ↪ 0.0)
```

这里有两种情况

1. 当缩放后的图片小于原始图片，再根据是原始图片宽小于缩放后的图片还是原始图片高小于缩放后的图片做不同的偏移操作
2. 当缩放后的图片大于原始图片，则直接计算

```

w_off_orig = math.floor(w_off_orig)
h_off_orig = math.floor(h_off_orig)

# Figure out top left coordinates.
w_off = 0.0
h_off = 0.0
w_off = float(w_off_orig / image_width)
h_off = float(h_off_orig / image_height)

sampled_bbox = bbox(w_off, h_off,
                    w_off + float(sample_bbox_size / image_width),
                    h_off + float(sample_bbox_size / image_height))
return sampled_bbox

```

最后就是根据图片大小归一化，并用 `bbox` 这个类进行包装，进行值的返回

12. 总结

这个算法在 VGG16 这个骨干网络上得到了很好的成绩，他分析原始 FPN 在人脸检测上的不适用，并以 LFPN 加以改进。引入 CPM 模块，结合了当时两个效果比较好的结构，一定程度上能提了点

Table 3: Context-sensitive Predict Module.

| Method | DSSD prediction module | SSH context module | CPM |
|------------|------------------------|--------------------|-------------|
| easy | 95.3 | 95.5 | 95.6 |
| mAP medium | 94.3 | 94.3 | 94.5 |
| hard | 88.2 | 88.4 | 88.5 |

Figure 9: image-20200627130752561

创新的 PyramidAnchor 给人脸检测带来了更丰富的上下文信息，帮助定位人脸。data-anchor-sampling 作为一种新的缩放采样方法也一定程度提高了模型性能

Table 4: Contrast results of the PyramidBox on WIDER FACE validation subset.

| Contribution | | Baseline | | | | | PyramidBox |
|-------------------------|--------|----------|------|------|------|------|-------------|
| lfpn? | | | ✓ | ✓ | ✓ | ✓ | ✓ |
| data-anchor-sampling? | | | | ✓ | ✓ | ✓ | ✓ |
| pyramid-anchors? | | | | | ✓ | ✓ | ✓ |
| context-predict-module? | | | | | | ✓ | ✓ |
| max-in-out? | | | | | | | ✓ |
| mAP | easy | 94.0 | 94.3 | 94.7 | 95.5 | 95.9 | 96.1 |
| | medium | 92.7 | 93.3 | 93.7 | 94.3 | 94.7 | 95.0 |
| | hard | 84.2 | 86.1 | 86.7 | 88.3 | 88.8 | 88.9 |

Figure 10: image-20200627130944291