Maix YOLO

1. YOLO 结构

K210 使用的 YOLO 版本为 YOLO v2，相较于 YOLO v1 在保留了高检测速度的同时，大幅度提高了定位精度。

> 🌈
>
> 具体描述可看这篇[文章](#)。

在基础模型（特征提取器）上，maix 采用 MobileNet V1这一轻量化网络来实现对图片信息的快速提取（通过牺牲一点准确率换取较小模型参数量，这实际上恰好适应嵌入式设备资源，算力有限的情形，与之类似网络的还有Tiny YOLO ，VGG16，ResNet18等）。

V1网络结构

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
5× Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
5× Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

在maix_train\train\detector\yolo\backend\utils\mobilenet_sipeed\mobilenet.py 文件里，可以看到对应层次结构的实现（网络截止到 `Avg Pool` ，输出特征图为7 X 7 X 1024）：

```python
    x = _conv_block(img_input, 32, alpha, strides=(2, 2))
    x = _depthwise_conv_block(x, 64, alpha, depth_multiplier, block_id=1)

    x = _depthwise_conv_block(x, 128, alpha, depth_multiplier,
                              strides=(2, 2), block_id=2)
    x = _depthwise_conv_block(x, 128, alpha, depth_multiplier, block_id=3)

    x = _depthwise_conv_block(x, 256, alpha, depth_multiplier,
                              strides=(2, 2), block_id=4)
    x = _depthwise_conv_block(x, 256, alpha, depth_multiplier, block_id=5)

    x = _depthwise_conv_block(x, 512, alpha, depth_multiplier,
                              strides=(2, 2), block_id=6)
    x = _depthwise_conv_block(x, 512, alpha, depth_multiplier, block_id=7)
    x = _depthwise_conv_block(x, 512, alpha, depth_multiplier, block_id=8)
    x = _depthwise_conv_block(x, 512, alpha, depth_multiplier, block_id=9)
    x = _depthwise_conv_block(x, 512, alpha, depth_multiplier, block_id=10)
    x = _depthwise_conv_block(x, 512, alpha, depth_multiplier, block_id=11)
    if total_strip_size == 32:
        x = _depthwise_conv_block(x, 1024, alpha, depth_multiplier,
                                  strides=(2, 2), block_id=12)
    elif total_strip_size == 16:
        x = _depthwise_conv_block(x, 1024, alpha, depth_multiplier,
                                  strides=(1, 1), block_id=12)
    else:
        raise Exception("total_strip_size not support, only support 16 and 32")
    x = _depthwise_conv_block(x, 1024, alpha, depth_multiplier, block_id=13)
```

_depthwise_conv_block 实现先 dw卷积再普通卷积：

```python
    if strides == (1, 1):
        x = inputs
    else:
        x = layers.ZeroPadding2D(((1, 1), (1, 1)),
                                 name='conv_pad_%d' % block_id)(inputs)
    x = layers.DepthwiseConv2D((3, 3),
                               padding='same' if strides == (1, 1) else 'valid',
                               depth_multiplier=depth_multiplier,
                               strides=strides,
                               use_bias=False,
                               name='conv_dw_%d' % block_id)(x)
    x = layers.BatchNormalization(
        axis=channel_axis, name='conv_dw_%d_bn' % block_id)(x)
    x = layers.ReLU(6., name='conv_dw_%d_relu' % block_id)(x)

    x = layers.Conv2D(pointwise_conv_filters, (1, 1),
                      padding='same',
                      use_bias=False,
                      strides=(1, 1),
                      name='conv_pw_%d' % block_id)(x)
    x = layers.BatchNormalization(axis=channel_axis,
                                  name='conv_pw_%d_bn' % block_id)(x)
    return layers.ReLU(6., name='conv_pw_%d_relu' % block_id)(x)
```

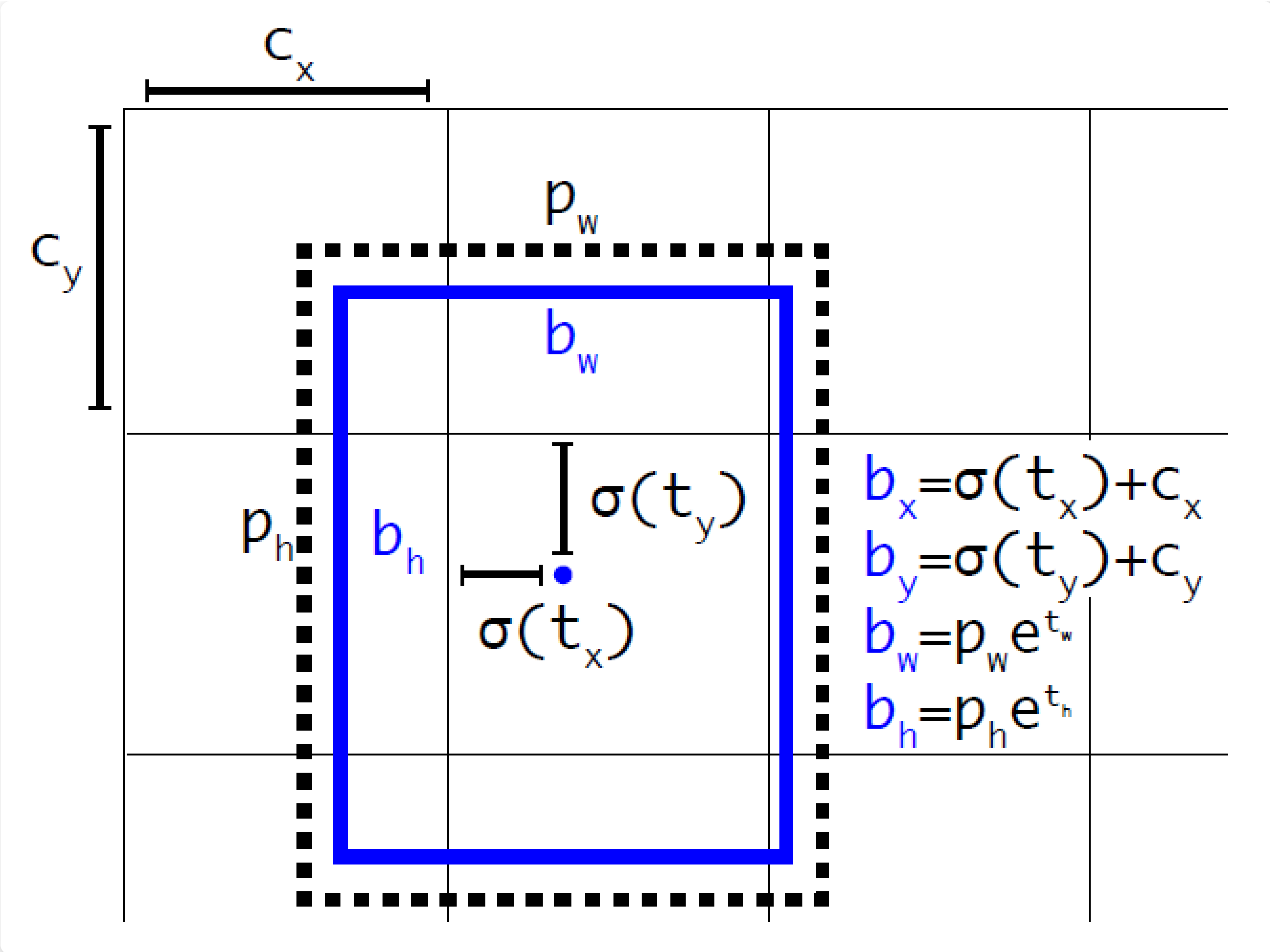
在 `maix_train\train\detector\yolo\backend\network.py` 里可以看到整个骨干网络的构建（feature_extractor + output_tensor）：

```python
class YoloNetwork(object):

    def __init__(self,
                 feature_extractor,
                 input_size,
                 nb_classes,
                 nb_box):
        from tensorflow.keras.models import Model
        from tensorflow.keras.layers import Reshape, Conv2D

        # 1. create full network
        grid_size = feature_extractor.get_output_size()

        # make the object detection layer
        output_tensor = Conv2D(nb_box * (4 + 1 + nb_classes), (1,1), strides=(1,1),
                               padding='same',
                               name='detection_layer_{}'.format(nb_box * (4 + 1 + nb_classes)),
                               kernel_initializer='lecun_normal')(feature_extractor.feature_extractor.output)
        output_tensor = Reshape((grid_size[0], grid_size[1], nb_box, 4 + 1 + nb_classes))(output_tensor)

        model = Model(feature_extractor.feature_extractor.input, output_tensor)

        self._norm = feature_extractor.normalize
        self._model = model
        self._model.summary()
        self._init_layer()
```

output_tensor 实际上就是在特征图上卷积做预测，输出通道数取决于 anchor boxes 数量以及检测的物品类别数。

在定位上，YOLOv2借鉴了RPN网络使用anchor boxes来预测边界框相对先验框的offsets。

$$
\begin{aligned}
b_x &= (\sigma(t_x) + c_x)/W \\
b_y &= (\sigma(t_y) + c_y)/H \\
b_w &= p_w e^{t_w}/W \\
b_h &= p_h e^{t_h}/H
\end{aligned}
$$



这部分对模型输出数据的解码可以参看 maix_train\train\detector\yolo\backend\decoder.py：（其中也包含非极大值抑制NMS，即去除相关联框）

```python
class YoloDecoder(object):

    def __init__(self,
                 anchors = [0.57273, 0.677385, 1.87446, 2.06253, 3.33843, 5.47434, 7.88282, 3.52778, 9.77052, 9.16828],
                 nms_threshold=0.2):
        self._anchors = anchors
        self._nms_threshold = nms_threshold

    def run(self, netout, obj_threshold=0.3):
        """Convert Yolo network output to bounding box

        # Args
            netout : 4d-array, shape of (grid_h, grid_w, num of boxes per grid, 5 + n_classes)
                YOLO neural network output array

        # Returns
            boxes : array, shape of (N, 4)
                coordinate scale is normalized [0, 1]
            probs : array, shape of (N, nb_classes)
        """
        grid_h, grid_w, nb_box = netout.shape[:3]

        boxes = []

        # decode the output by the network
        netout[..., 4]  = _sigmoid(netout[..., 4])
        netout[..., 5:] = netout[..., 4][..., np.newaxis] * _softmax(netout[..., 5:])
        netout[..., 5:] *= netout[..., 5:] > obj_threshold

        for row in range(grid_h):
            for col in range(grid_w):
                for b in range(nb_box):
                    # from 4th element onwards are confidence and class classes
                    classes = netout[row,col,b,5:]

                    if np.sum(classes) > 0:
                        # first 4 elements are x, y, w, and h
                        x, y, w, h = netout[row,col,b,:4]

                        x = (col + _sigmoid(x)) / grid_w # center position, unit: image width
                        y = (row + _sigmoid(y)) / grid_h # center position, unit: image height
                        w = self._anchors[2 * b + 0] * np.exp(w) / grid_w # unit: image width
                        h = self._anchors[2 * b + 1] * np.exp(h) / grid_h # unit: image height
                        confidence = netout[row,col,b,4]
                        box = BoundBox(x, y, w, h, confidence, classes)
                        boxes.append(box)

        boxes = nms_boxes(boxes, len(classes), self._nms_threshold, obj_threshold)
        boxes, probs = boxes_to_array(boxes)
        return boxes, probs
```

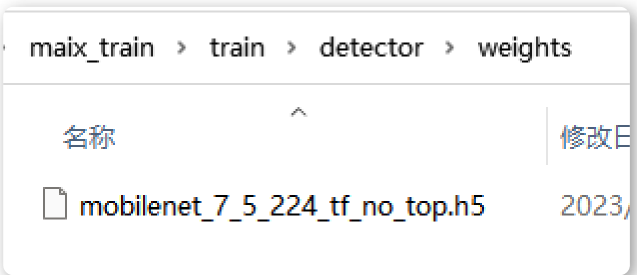
而在anchor boxes 值的选取上，YOLOv2并没有使用预设的纵横比和尺度的组合（Faster R-CNN 定义三组纵横比 `ratio = [0.5,1,2]` 和三种尺度 `scale = [8,16,32]` ，可以组合出9种不同的形状和大小的边框），而是使用 `k-means` 聚类的方法，从训练集中学习得到不同的Anchor。

在 `maix_train\train\detector\__init__.py` 里有计算anchors的具体实现：

```python
    def get_anchors(self, bboxes_in, input_shape=(224, 224), clusters = 5, strip_size = 32):
        '''
            @input_shape tuple (h, w)
            @bboxes_in format: [ [[xmin,ymin, xmax, ymax, label],], ]
                        value range: x [0, w], y [0, h]
            @return anchors, format: 10 value tuple
        '''
        w = input_shape[1]
        h = input_shape[0]
        # TODO: add position to iou, not only box size
        bboxes = []
        for items in bboxes_in:
            for bbox in items:
                bboxes.append(( (bbox[2] - bbox[0])/w, (bbox[3] - bbox[1])/h ))
        bboxes = np.array(bboxes)
        self.log.i(f"bboxes num: {len(bboxes)}, first bbox: {bboxes[0]}")
        out = kmeans.kmeans(bboxes, k=clusters)
        iou = kmeans.avg_iou(bboxes, out) * 100
        self.log.i("bbox accuracy(IOU): {:.2f}%".format(iou))
        self.log.i("bound boxes: {}".format( ",".join("({:f},{:.2f})".format(item[0] * w, item[1] * h) for item in out) ))
        for i, wh in enumerate(out):
            out[i][0] = wh[0]*w/strip_size
            out[i][1] = wh[1]*h/strip_size
        anchors = list(out.flatten())
        self.log.i(f"anchors: {anchors}")
        ratios = np.around(out[:, 0] / out[:, 1], decimals=2).tolist()
        self.log.i("w/h ratios: {}".format(sorted(ratios)))
        return anchors
```

2. Train

为了加快训练速率，提高模型的泛化性，maix 的训练方式为迁移学习，在 maix_train\train\detector\weights 文件夹下，已提供了特征提取器mobilenet 的预训练参数文件。



因此实际训练时只需微调（FineTune）网络，即根据检测物品类别修改输出层（这一点在网络最后output_tensor 上有体现）等。