YOLOv3
: An Incremental Improvement (2018)

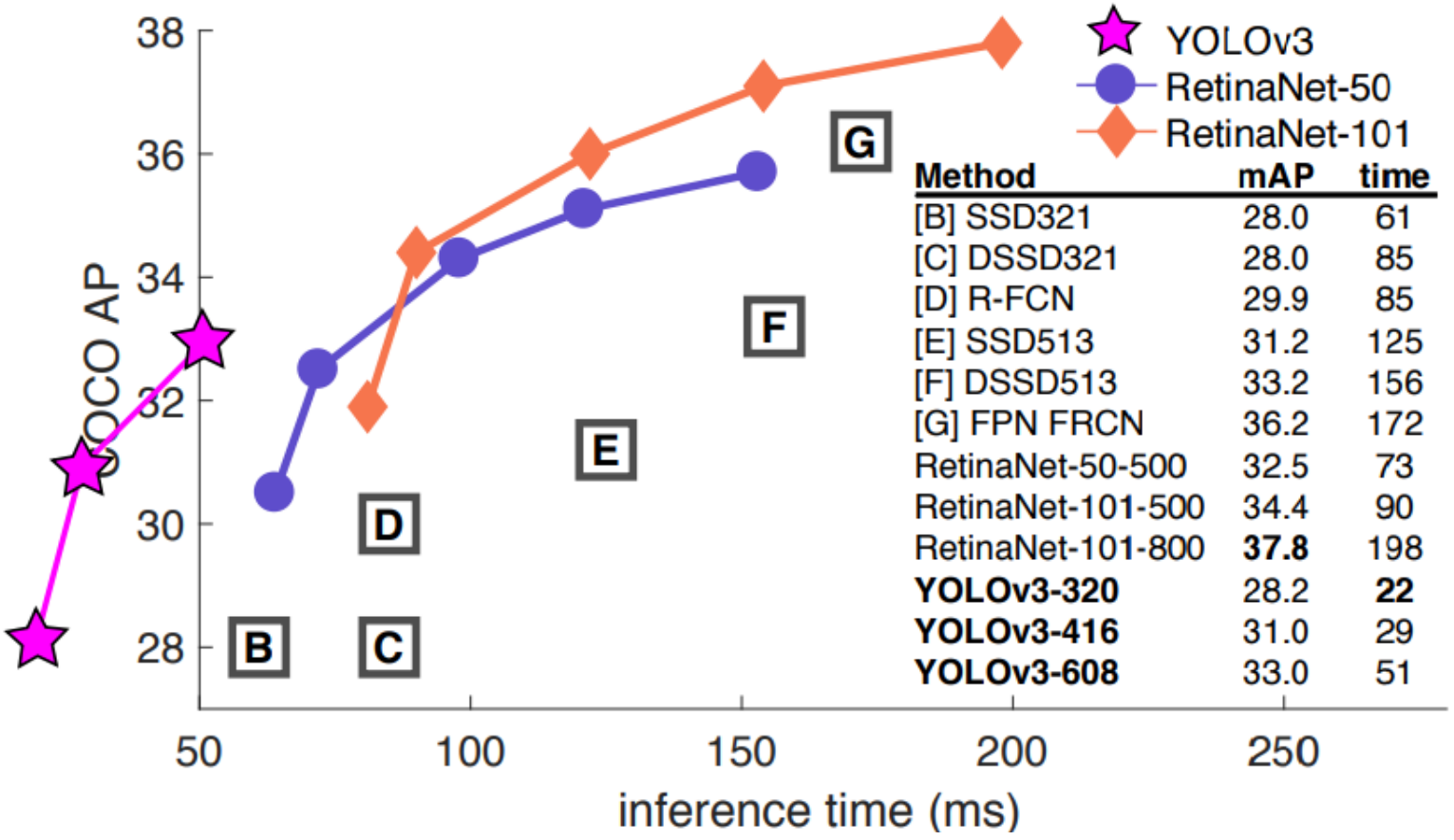
Seho Kim

https://arxiv.org/pdf/1804.02767.pdf

Introduction

- 320x320 YOLOv3 runs in 22ms at 28.2 mAP

- TECH REPORT



Method	mAP	time
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

Bounding Box Prediction

- Following YOLO9000
  - Using dimension clusters as anchor boxes.
  - Predicts 4 coordinates for each bounding box

$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$

  - Use SSE(Sum of Squared Error) loss

Bounding Box Prediction

- YOLOv3
  - Predicts an objectness score for each bounding box using logistic regression
  - Best IOU with ground truth → 1, ignore the others
  - One bounding box prior for each ground truth object (if it is not assigned, no loss coordinate or class predictions, only objectness)

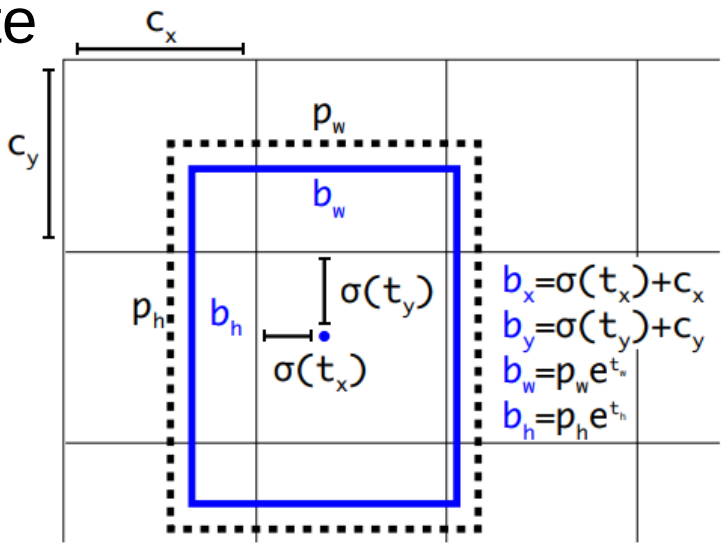


Figure 2. **Bounding boxes with dimension priors and location prediction.** We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function. This figure blatantly self-plagiarized from [15].

Class Prediction

- Softmax is not used

  → use independent logistic classifiers

- During training, use binary cross-entropy loss for predictions

  Helps for more complex domains; multilabel

  (ex. Open Images Dataset; many overlapping labels)

Predictions Across Scales

- YOLOv3 predicts boxes at 3 different scales
  - Using similar concept to feature pyramid networks(FPN)
  - Predict 3 boxes at each scale

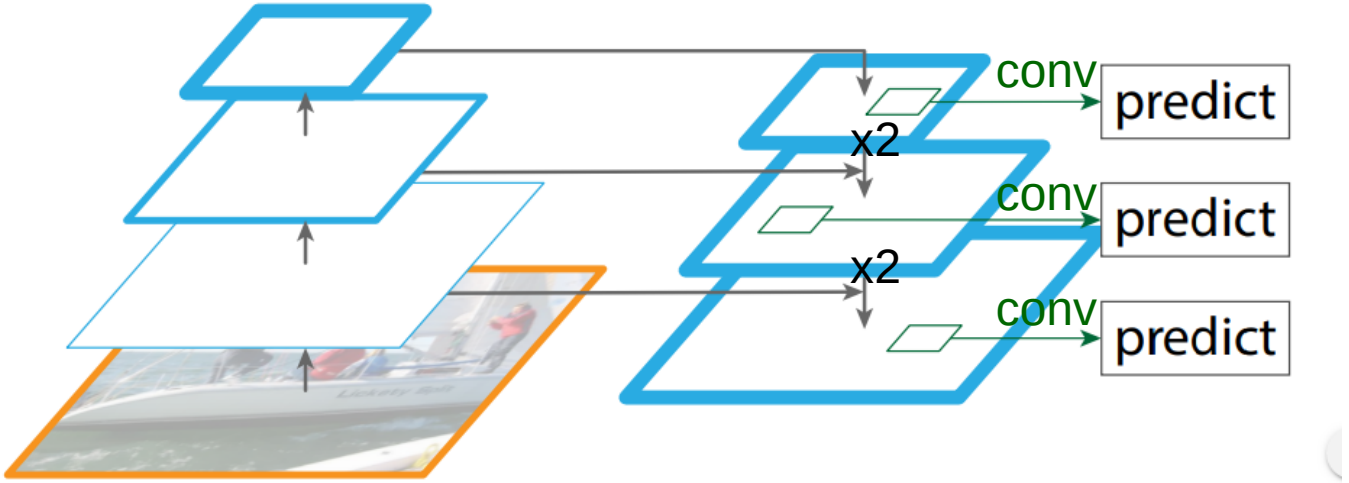
    N x N x [3 * (4 + 1 + 80)]

    ; Grid N, 3 Boxes, 4 bounding box offsets,

    1 objectness prediction, 80 class

Predictions Across Scales

- YOLOv3 predicts boxes at 3 different scales
  - Take the feature map from 2 layers previous
    and upsample it by 2x
  - Also Take a feature map from earlier in the network
    (finer-grained information)
    and merge it with upsampled features
    (to get more meaningful semantic information)
  - Add a few more convolutional layers



Predictions Across Scales

- YOLOv3 predicts boxes at 3 different scales
  - Still use k-means clustering to determine bounding box priors
  - Sort of chose 9 clusters and 3 scales arbitrarily and then divide up the clusters evenly across scales

    9 clusters = 3 boxes * 3 scales

    (ex. COCO dataset 9 clusters: (10×13),(16×30),(33×23), (30×61),(62×45),(59×119),(116 × 90),(156 × 198), (373 × 326))

Feature Extractor

- Darknet-53
  - Use successive 3x3 and 1x1 convolutional layers
  - Shortcut connections

    ; significantly larger

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	**171**
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	**77.6**	**93.8**	29.4	1090	37
Darknet-53	77.2	**93.8**	18.7	**1457**	78

Training

- Full image
- With no hard negative mining...
- Multi-scale training, data augmentation, batch normalization...

How We Do

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [5]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [8]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [6]	Inception-ResNet-v2 [21]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [20]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [15]	DarkNet-19 [15]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [11, 3]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [3]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [9]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [9]	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2
YOLOv3 608×608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

Things We Tried That Didn’t Work

- Anchor box x, y offset predictions
- Linear x, y predictions instead of logistic
- Focal loss
- Dual IOU thresholds and truth assignment

What This All Means

```
242  class YoloNetV3(nn.Module):
243
244      def __init__(self, nms=False, post=True):
245          super(YoloNetV3, self).__init__()
246          self.darknet = DarkNet53BackBone()
247          self.yolo_tail = YoloNetTail()
248          self.nms = nms
249          self._post_process = post
250
251      def forward(self, x):
252          tmp1, tmp2, tmp3 = self.darknet(x)
253          out1, out2, out3 = self.yolo_tail(tmp1, tmp2, tmp3)
254          out = torch.cat((out1, out2, out3), 1)
255          logging.debug("The dimension of the output before nms is {}".format(out.size()))
256          return out
```

https://github.com/westerndigitalcorporation/YOLOv3-in-PyTorch

```
38  class ConvLayer(nn.Module):
39      """Basic 'conv' layer, including:
40       A Conv2D layer with desired channels and kernel size,
41       A batch-norm layer,
42       and A leakyReLu layer with neg_slope of 0.1.
43       (Didn't find too much resource what neg_slope really is.
44       By looking at the darknet source code, it is confirmed the neg_slope=0.1.
45       Ref: https://github.com/pjreddie/darknet/blob/master/src/activations.h)
46       Please note here we distinguish between Conv2D layer and Conv layer."""
47
48      def __init__(self, in_channels, out_channels, kernel_size, stride=1, lrelu_neg_slope=0.1):
49          super(ConvLayer, self).__init__()
50          padding = (kernel_size - 1) // 2
51          self.conv = nn.Conv2d(in_channels, out_channels, kernel_size, stride, padding, bias=False)
52          self.bn = nn.BatchNorm2d(out_channels)
53          self.lrelu = nn.LeakyReLU(negative_slope=lrelu_neg_slope)
54
55      def forward(self, x):
56          out = self.conv(x)
57          out = self.bn(out)
58          out = self.lrelu(out)
59
60          return out
```

https://github.com/westerndigitalcorporation/YOLOv3-in-PyTorch

```
189  class DarkNet53BackBone(nn.Module):
190
191      def __init__(self):
192          super(DarkNet53BackBone, self).__init__()
193          self.conv1 = ConvLayer(3, 32, 3)
194          self.cr_block1 = make_conv_and_res_block(32, 64, 1)
195          self.cr_block2 = make_conv_and_res_block(64, 128, 2)
196          self.cr_block3 = make_conv_and_res_block(128, 256, 8)
197          self.cr_block4 = make_conv_and_res_block(256, 512, 8)
198          self.cr_block5 = make_conv_and_res_block(512, 1024, 4)
199
200      def forward(self, x):
201          tmp = self.conv1(x)
202          tmp = self.cr_block1(tmp)
203          tmp = self.cr_block2(tmp)
204          out3 = self.cr_block3(tmp)
205          out2 = self.cr_block4(out3)
206          out1 = self.cr_block5(out2)
207
208          return out1, out2, out3
209
```

https://github.com/westerndigitalcorporation/YOLOv3-in-PyTorch

```python
class ResBlock(nn.Module):
    """The basic residual block used in YoloV3.
    Each ResBlock consists of two ConvLayers and the input is added to the final output.
    In YoloV3 paper, the first convLayer has half of the number of the filters as much as the second convLayer.
    The first convLayer has filter size of 1x1 and the second one has the filter size of 3x3.
    """

    def __init__(self, in_channels):
        super(ResBlock, self).__init__()
        assert in_channels % 2 == 0  # ensure the in_channels is an even number.
        half_in_channels = in_channels // 2
        self.conv1 = ConvLayer(in_channels, half_in_channels, 1)
        self.conv2 = ConvLayer(half_in_channels, in_channels, 3)

    def forward(self, x):
        residual = x
        out = self.conv1(x)
        out = self.conv2(out)
        out += residual

        return out


def make_conv_and_res_block(in_channels, out_channels, res_repeat):
    """In Darknet 53 backbone, there is usually one Conv Layer followed by some ResBlock.
    This function will make that.
    The Conv layers always have 3x3 filters with stride=2.
    The number of the filters in Conv layer is the same as the out channels of the ResBlock"""
    model = nn.Sequential()
    model.add_module('conv', ConvLayer(in_channels, out_channels, 3, stride=2))
    for idx in range(res_repeat):
        model.add_module('res{}'.format(idx), ResBlock(out_channels))
    return model
```

https://github.com/westerndigitalcorporation/YOLOv3-in-PyTorch

```
class YoloNetTail(nn.Module):

    """The tail side of the YoloNet.
    It will take the result from DarkNet53BackBone and do some upsampling and concatenation.
    It will finally output the detection result.
    Assembling YoloNetTail and DarkNet53BackBone will give you final result"""

    def __init__(self):
        super(YoloNetTail, self).__init__()
        self.detect1 = DetectionBlock(1024, 1024, 'l', 32)
        self.conv1 = ConvLayer(512, 256, 1)
        self.detect2 = DetectionBlock(768, 512, 'm', 16)
        self.conv2 = ConvLayer(256, 128, 1)
        self.detect3 = DetectionBlock(384, 256, 's', 8)

    def forward(self, x1, x2, x3):
        out1 = self.detect1(x1)
        branch1 = self.detect1.branch
        tmp = self.conv1(branch1)
        tmp = F.interpolate(tmp, scale_factor=2)
        tmp = torch.cat((tmp, x2), 1)
        out2 = self.detect2(tmp)
        branch2 = self.detect2.branch
        tmp = self.conv2(branch2)
        tmp = F.interpolate(tmp, scale_factor=2)
        tmp = torch.cat((tmp, x3), 1)
        out3 = self.detect3(tmp)

        return out1, out2, out3
```

https://github.com/westerndigitalcorporation/YOLOv3-in-PyTorch

```
152 class DetectionBlock(nn.Module):
153     """The DetectionBlock contains:
154     Six ConvLayers, 1 Conv2D Layer and 1 YoloLayer.
155     The first 6 ConvLayers are formed the following way:
156     1x1xn, 3x3x2n, 1x1xn, 3x3x2n, 1x1xn, 3x3x2n,
157     The Conv2D layer is 1x1x255.
158     Some block will have branch after the fifth ConvLayer.
159     The input channel is arbitrary (in_channels)
160     out_channels = n
161     """
162 
163     def __init__(self, in_channels, out_channels, scale, stride):
164         super(DetectionBlock, self).__init__()
165         assert out_channels % 2 == 0  #assert out_channels is an even number
166         half_out_channels = out_channels // 2
167         self.conv1 = ConvLayer(in_channels, half_out_channels, 1)
168         self.conv2 = ConvLayer(half_out_channels, out_channels, 3)
169         self.conv3 = ConvLayer(out_channels, half_out_channels, 1)
170         self.conv4 = ConvLayer(half_out_channels, out_channels, 3)
171         self.conv5 = ConvLayer(out_channels, half_out_channels, 1)
172         self.conv6 = ConvLayer(half_out_channels, out_channels, 3)
173         self.conv7 = nn.Conv2d(out_channels, LAST_LAYER_DIM, 1, bias=True)
174         self.yolo = YoloLayer(scale, stride)
175 
176     def forward(self, x):
177         tmp = self.conv1(x)
178         tmp = self.conv2(tmp)
179         tmp = self.conv3(tmp)
180         tmp = self.conv4(tmp)
181         self.branch = self.conv5(tmp)
182         tmp = self.conv6(self.branch)
183         tmp = self.conv7(tmp)
184         out = self.yolo(tmp)
185 
186         return out
```

https://github.com/westerndigitalcorporation/YOLOv3-in-PyTorch

```python
class YoloLayer(nn.Module):

    def __init__(self, scale, stride):
        super(YoloLayer, self).__init__()
        if scale == 's':
            idx = (0, 1, 2)
        elif scale == 'm':
            idx = (3, 4, 5)
        elif scale == 'l':
            idx = (6, 7, 8)
        else:
            idx = None
        self.anchors = torch.tensor([ANCHORS[i] for i in idx])
        self.stride = stride

    def forward(self, x):
        num_batch = x.size(0)
        num_grid = x.size(2)

        if self.training:
            output_raw = x.view(num_batch,
                                NUM_ANCHORS_PER_SCALE,
                                NUM_ATTRIB,
                                num_grid,
                                num_grid).permute(0, 1, 3, 4, 2).contiguous().view(num_batch, -1, NUM_ATTRIB)
            return output_raw
        else:
            prediction_raw = x.view(num_batch,
                                    NUM_ANCHORS_PER_SCALE,
                                    NUM_ATTRIB,
                                    num_grid,
                                    num_grid).permute(0, 1, 3, 4, 2).contiguous()

            self.anchors = self.anchors.to(x.device).float()
            # Calculate offsets for each grid
            grid_tensor = torch.arange(num_grid, dtype=torch.float, device=x.device).repeat(num_grid, 1)
            grid_x = grid_tensor.view([1, 1, num_grid, num_grid])
            grid_y = grid_tensor.t().view([1, 1, num_grid, num_grid])
            anchor_w = self.anchors[:, 0:1].view((1, -1, 1, 1))
            anchor_h = self.anchors[:, 1:2].view((1, -1, 1, 1))

            # Get outputs
            x_center_pred = (torch.sigmoid(prediction_raw[..., 0]) + grid_x) * self.stride # Center x
            y_center_pred = (torch.sigmoid(prediction_raw[..., 1]) + grid_y) * self.stride  # Center y
            w_pred = torch.exp(prediction_raw[..., 2]) * anchor_w  # Width
            h_pred = torch.exp(prediction_raw[..., 3]) * anchor_h  # Height
            bbox_pred = torch.stack((x_center_pred, y_center_pred, w_pred, h_pred), dim=4).view((num_batch, -1, 4)) #cxcywh
            conf_pred = torch.sigmoid(prediction_raw[..., 4]).view(num_batch, -1, 1)  # Conf
            cls_pred = torch.sigmoid(prediction_raw[..., 5:]).view(num_batch, -1, NUM_CLASSES)  # Cls pred one-hot.

            output = torch.cat((bbox_pred, conf_pred, cls_pred), -1)
            return output
```

https://github.com/westerndigitalcorporation/YOLOv3-in-PyTorch