Generating Positive Bounding Boxes for Balanced Training of Object Detectors

Kemal Oksuz, Baris Can Cam, Emre Akbas*, Sinan Kalkan*
Department of Computer Engineering
Middle East Technical University, Ankara, Turkey
{kemal.oksuz, can.cam, eakbas, skalkan}@metu.edu.tr

Abstract

In this document, we provide supplementary material that were omitted in the submitted manuscript due to space constraints. This supplementary material includes the proofs for the theorems, some more details on the BB generator including derivations for equations 4 and 5 in the paper and computing the feasible space in the bottom right part, and finally some implementation details. As for implementation details, we discuss how the pRoI generator is integrated into the training, how pRoI generator uses BB generator as a subroutine and configurations of BB sources.

1. The Properties of IoU(B, B̄)

Following upon the notation in Section 3 of the paper, we introduce the following properties. For clarity we assume that intersection of two boxes is greater than 0 and the last pixel is not taken into account (i.e. instead of A(B) = (x_2 - x_1 + 1), we adopted A(B) = (x_2 - x_1)).

Theorem 1. IoU(B, B̄) is scale-invariant.

Proof. Assume that k_x > 0 and k_y > 0 are the scaling factors in the x and y axes respectively and B_s, B̄_s are the scaled boxes. We show that IoU(B_s, B̄_s) = IoU(B, B̄) as follows:

\[
\text{IoU}(B_s, B̄_s) = \frac{I(B_s, B̄_s)}{A(B_s) + A(B̄_s) - I(B_s, B_s)}
\]

(1)

\[
= \frac{(\min(k \times x_2, k \times x_2) - \max(k \times x_1, k \times x_1)) \times (\min(k \times y_2, k \times y_2) - \max(k \times y_1, k \times y_1))}{(k \times x_2 - k \times x_1) \times (k \times y_2 - k \times y_1) + (k \times x_2 - k \times x_1) \times (k \times y_2 - k \times y_1) - I(B_s, B̄_s)}
\]

(2)

\[
= k_x (\min(y_2, y_2) - \max(y_1, y_1)) \times k_y (\min(x_2, x_2) - \max(x_1, x_1))
\]

(3)

\[
= k_x k_y (x_2 - x_1) \times (y_2 - y_1) + k_x k_y (x_2 - x_1) \times (y_2 - y_1) - k_x k_y I(B, B̄)
\]

(4)

\[
= k_x k_y (x_2 - x_1) \times (y_2 - y_1) + k_x k_y (x_2 - x_1) \times (y_2 - y_1) - k_x k_y I(B, B̄)
\]

(5)

\[
= \text{IoU}(B, B̄)
\]

(6)

Eq. 1 defines the IoU and Eq. 2 replaces area and intersection definitions. In Eq. 3, we use the property that multiplying by a positive scalar does not change minimum and maximum of two numbers. Eq. 4 incorporates the intersection definition. Eq. 5 gets the denominator in the k^k parenthesis, which simplifies the term to the definition of IoU(B, B̄).

Theorem 2. IoU(B, B̄) is translation-invariant.

Proof. Assuming that k_x \in \mathbb{R} and k_y \in \mathbb{R} are the perturbation in the x and y axis respectively and B_t, B̄_t are the perturbed boxes. We show that IoU(B_t, B̄_t) = IoU(B, B̄) as follows:

\[
\text{IoU}(B_t, B̄_t) = \frac{I(B_t, B̄_t)}{A(B_t) + A(B̄_t) - I(B_t, B_t)}
\]

(7)

\[
= \frac{(\min(k \times x_2, k \times x_2) - \max(k \times x_1, k \times x_1)) \times (\min(k \times y_2, k \times y_2) - \max(k \times y_1, k \times y_1))}{(k \times x_2 - k \times x_1) \times (k \times y_2 - k \times y_1) + (k \times x_2 - k \times x_1) \times (k \times y_2 - k \times y_1) - I(B_t, B̄_t)}
\]

(8)

\[
= k_x (\min(y_2, y_2) - \max(y_1, y_1)) \times k_y (\min(x_2, x_2) - \max(x_1, x_1))
\]

(9)

\[
= k_x k_y (x_2 - x_1) \times (y_2 - y_1) + k_x k_y (x_2 - x_1) \times (y_2 - y_1) - k_x k_y I(B, B̄)
\]

(10)

\[
= k_x k_y (x_2 - x_1) \times (y_2 - y_1) + k_x k_y (x_2 - x_1) \times (y_2 - y_1) - k_x k_y I(B, B̄)
\]

(11)

\[
= \text{IoU}(B, B̄)
\]

(12)

*Equal contribution for senior authorship.
\[ \text{IoU}(B_1, B_2) = \frac{I(B_1, B_2)}{A(B_1) + A(B_2) - I(B_1, B_2)} \]

\[ = \frac{(\min (x_2 + k_x, x_2 + k_x) - \max (x_1 + k_x, x_1 + k_x)) \times (\min (y_2 + k_y, y_2 + k_y) - \max (y_1 + k_y, y_1 + k_y))}{((x_2 + k_x) - (x_1 + k_x)) \times ((y_2 + k_y) - (y_1 + k_y)) + ((x_2 + k_x) - (x_1 + k_x)) \times ((y_2 + k_y) - (y_1 + k_y) - I(B_1, B_2))} \]

\[ = \frac{(x_2 + k_x - x_1 - k_x) \times (y_2 + k_y - y_1 - k_y) + (x_2 + k_x - x_1 - k_x) \times (y_2 + k_y - y_1 - k_y) - I(B_1, B_2)}{(x_2 - x_1) \times (y_2 - y_1) + (x_2 - x_1) \times (y_2 - y_1) - I(B_1, B_2)} \]

\[ = \frac{I(B_1, B_2)}{A(B_1) + A(B_2) - I(B_1, B_2)} \]

Again, Eq. 8 replaces area and intersection definitions in the IoU definition. In Eq. 9, we use the property that adding a scalar to numbers adds a scalar to the minimum and maximum of two numbers. In Eq. 10, constants cancel each other and Eq. 11 replaces area and intersection for the IoU function, which simplifies to the definition of IoU(B, B).

2. Details of the Bounding Box Generator

In this section we present the derivation of the Equation 4 and 5, and explain the \text{findBRFeasibleSpace}(B, T, \text{TL}(\hat{B})) function.

2.1. \text{findTLFeasibleSpace}(B, T) function

Here, we derive Equation 4 and 5 in the paper, and present the equations for the top-left space..

\[ \text{IoU}(B, \hat{B}) = \frac{I(B, \hat{B})}{A(B) + A(\hat{B}) - I(B, \hat{B})} \]

\[ = \frac{(\min (\bar{x}_2, x_2) - \max (\bar{x}_1, x_1)) \times (\min (\bar{y}_2, y_2) - \max (\bar{y}_1, y_1))}{(x_2 - x_1) \times (y_2 - y_1) + (x_2 - x_1) \times (y_2 - y_1) - I(B, \hat{B})} \]

\[ \Rightarrow T = \frac{(x_2 - x_1) \times (y_2 - y_1) + (x_2 - x_{\max}) \times (y_2 - y_1) - (x_2 - x_{\max}) \times (y_2 - y_1)}{(x_2 - x_1) \times (y_2 - y_1) \times T} \]

\[ = (x_2 - x_1) \times (y_2 - y_1) \times T \]

\[ \Rightarrow x'_{\max} = x_2 - \frac{(x_2 - x_1) \times (y_2 - y_1) \times T}{(y_2 - y_1)} \]

\[ \Rightarrow x'_{\max} = x_2 - (x_2 - x_1) \times T \]

In order to derive Equation 4 depicting \( x'_{\max} \), we bound the \( x \) coordinate first. It is obvious that \( x'_{\min} = x_1 \) due to the boundary of Region I. For \( x'_{\max} \), we know that \( \bar{y}_1 = y_1 \) again thanks to the region boundary. Therefore, since we have only one unknown, \( x'_{\max} \), we use Eq. 12 to determine its value.

\[ \text{IoU}(B, \hat{B}) = \frac{I(B, \hat{B})}{A(B) + A(\hat{B}) - I(B, \hat{B})} \]

\[ \Rightarrow T \times (x_2 - \bar{x}_1) \times (y_2 - \bar{y}_1) \]

\[ = I(B, \hat{B}) + T \times I(B, \hat{B}) - T \times A(B) \]

\[ \Rightarrow \bar{y}_1 = y_2 - \frac{I(B, \hat{B}) + I(B, \hat{B}) - A(B)}{(x_2 - x_1)} \]

Now since we know the values of \( \bar{x}_1 \) based on the bounds, we can derive the Equation 5 (in the paper) for any \( \bar{y}_1 \) value in equations by moving within bounds. Since \( I(B, \hat{B}) = (x_2 - \bar{x}_1) \times (y_2 - y_1) \), it does not rely on \( \bar{y}_1 \) and we directly use \( I(B, \hat{B}) \) in the following equations:
is trained with the gradients back-propagated from the protector (i.e. R-CNN [2], R-FCN [1]). The feature extractor (i.e. RPN [4]), and the third network, which is the object detector (i.e. R-CNN [2], R-FCN [1]). The feature extractor is trained with the gradients back-propagated from the proposal generator and the object detector. The proposal generator is trained by a subset of the anchor-ground truth combinations (chosen by Sample Anchors to Train RPN in Figure 1) and a subset of these RPN proposals (i.e. RoIs) (chosen by Sample RoIs to Train R-CNN in Figure 1) are fed into the R-CNN after a series of operations including NMS and RoI Pooling that do not include learnable parameters. Finally, the loss is back-propagated through the entire network to update the parameters. However, the RoIs from the RPN is limited in number and diversity, which can impact the analysis and training. To address this, pRoI generator aims to generate RoIs with any desired property and in any number. Note that the gradients can also be back-propagated to the feature extractor as in the conventional training (i.e. RPN) since positive RoI Generator uses ground truths to generate an RoI in a similar manner to the conventional training, but differently it can generate boxes with the desired properties. During training, only for positive RoIs, pRoI Generator does not use the modules that are under the transparent red rectangle in Figure 1. However, during test time, our method follows the conventional approach, namely the RoIs from RPN are used due to the fact that no ground truth information is available during testing.

### 3. Implementation Details

#### 3.1. Integrating pRoI Generator into the Training

The training of the two-stage object detectors involves 3 different networks as shown in Fig. 1. The first network is the feature extractor (i.e. ResNet[3]) which presents the base features to the second network, the proposal generator (i.e. RPN [4]), and the third network, which is the object detector (i.e. R-CNN [2], R-FCN [1]). The feature extractor is trained with the gradients back-propagated from the proposal generator and the object detector. The proposal generator is trained by a subset of the anchor-ground truth combinations (chosen by Sample Anchors to Train RPN in Figure 1) and a subset of these RPN proposals (i.e. RoIs) (chosen by Sample RoIs to Train R-CNN in Figure 1) are fed into the R-CNN after a series of operations including NMS and RoI Pooling that do not include learnable parameters. Finally, the loss is back-propagated through the entire network to update the parameters. However, the RoIs from the RPN is limited in number and diversity, which can impact the analysis and training. To address this, pRoI generator aims to generate RoIs with any desired property and in any number. Note that the gradients can also be back-propagated to the feature extractor as in the conventional training (i.e. RPN) since positive RoI Generator uses ground truths to generate an RoI in a similar manner to the conventional training, but differently it can generate boxes with the desired properties. During training, only for positive RoIs, pRoI Generator does not use the modules that are under the transparent red rectangle in Figure 1. However, during test time, our method follows the conventional approach, namely the RoIs from RPN are used due to the fact that no ground truth information is available during testing.

#### 3.2. Connection Between genRoIs() and generateBB()

As described in the text, generateBB() is a low-level function and any approach uses generated BBs approach is to rely on this function. That’s why it is a subroutine of genRoIs(). The main idea in our implementation
Table 3: Bottom-right space equations.

<table>
<thead>
<tr>
<th>Region</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>( \bar{x}_2 = \bar{x}_1 + \frac{I(B,\bar{B}) - A(B) + I(B,\bar{B})}{T} )</td>
</tr>
<tr>
<td>II</td>
<td>( \bar{y}_2 = \bar{y}_1 + \frac{I(B,\bar{B}) - A(B) + I(B,\bar{B})}{T} )</td>
</tr>
<tr>
<td>III</td>
<td>( \bar{x}_2 = \frac{T \times A(B) + \alpha \times T \times (\beta - \beta) + \alpha \times (\beta - \beta) - T \times \bar{x}_1 \times (\bar{y}_2 - \bar{y}_1)}{(T+1) \times (\beta - \beta) - T \times (\bar{y}_2 - \bar{y}_1)} )</td>
</tr>
<tr>
<td>IV</td>
<td>( \bar{y}_2 = \frac{T \times A(B) + \beta \times T \times (\alpha - \alpha) + \beta \times (\alpha - \alpha) - T \times \bar{y}_1 \times (\bar{x}_2 - \bar{x}_1)}{(T+1) \times (\alpha - \alpha) - T \times (\bar{x}_2 - \bar{x}_1)} )</td>
</tr>
</tbody>
</table>

Figure 1: Conventional Faster R-CNN Training and our modification. During training, Positive RPN RoIs are not utilized and thus the modules presented under the large red rectangle are not used for positive RoIs. These RoIs are generated by the Positive RoI Generator shown in yellow box.

is to generate bounding boxes by iteratively calling the `generateBB()` for `RoINum` times.

Apart from `RoINum`, the number of RoIs to be generated, there are two main input sets to the `genRoIs` function. Firstly, `GTs` and `perGTRoI` together have the information about the box coordinates and the number of RoIs to be generated from each ground truth box. Therefore, for \( i \)-th ground truth box \( (GTs_i) \), we call `generateBB()` function for `perGTRoI_i` times. And the second set of input comprises \( \psi_{IoU} \) and \( W_{IoU} \), which together have information about the weights of each IoU interval. Therefore, for determining an IoU for each box, we first generate `perGTRoI` number of samples of IoU intervals using the multinomial distribution defined in \( W_{IoU} \) and then, for each resulting interval, we again sample uniformly an IoU within its limits. These IoUs are clipped from 0.95 in order to prevent the problems arising from the precision problem in the `samplePolygon()` acceptance process. This sampling strategy distributes the input IoUs over an interval evenly. Finally, we randomly shuffle this set of IoUs and associate them to the ground truths, which completes the generation of the ground truth and desired IoU pairs as the input of the `generateBB()` function.
Table 4: The configurations of $W_{IoU}$ for the different tables in the paper.

<table>
<thead>
<tr>
<th>Table</th>
<th>Role Source</th>
<th>$IoU = 0.5$</th>
<th>$IoU = 0.6$</th>
<th>$IoU = 0.7$</th>
<th>$IoU = 0.8$</th>
<th>$IoU = 0.9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Right Skew</td>
<td>0.33</td>
<td>0.33</td>
<td>0.30</td>
<td>0.20</td>
<td>0.95</td>
</tr>
<tr>
<td>1</td>
<td>Left Skew</td>
<td>0.73</td>
<td>0.12</td>
<td>0.15</td>
<td>0.05</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>Balanced, $IoU = 0.5$</td>
<td>0.33</td>
<td>0.17</td>
<td>0.16</td>
<td>0.17</td>
<td>0.15</td>
</tr>
<tr>
<td>4</td>
<td>Balanced, $IoU = 0.6$</td>
<td>0</td>
<td>0.38</td>
<td>0.20</td>
<td>0.22</td>
<td>0.20</td>
</tr>
<tr>
<td>4</td>
<td>Balanced, $IoU = 0.7$</td>
<td>0</td>
<td>0</td>
<td>0.48</td>
<td>0.25</td>
<td>0.27</td>
</tr>
<tr>
<td>4</td>
<td>Balanced, $IoU = 0.8$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.64</td>
<td>0.36</td>
</tr>
<tr>
<td>4</td>
<td>Balanced, $IoU = 0.9$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

3.3. Configurations of $W_{IoU}$

The configurations of the $W_{IoU}$ (i.e. the distribution over $\psi_{IoU} = [0.5, 0.6, 0.7, 0.8, 0.9]$) used for the experiments is shown in Table 4.

References