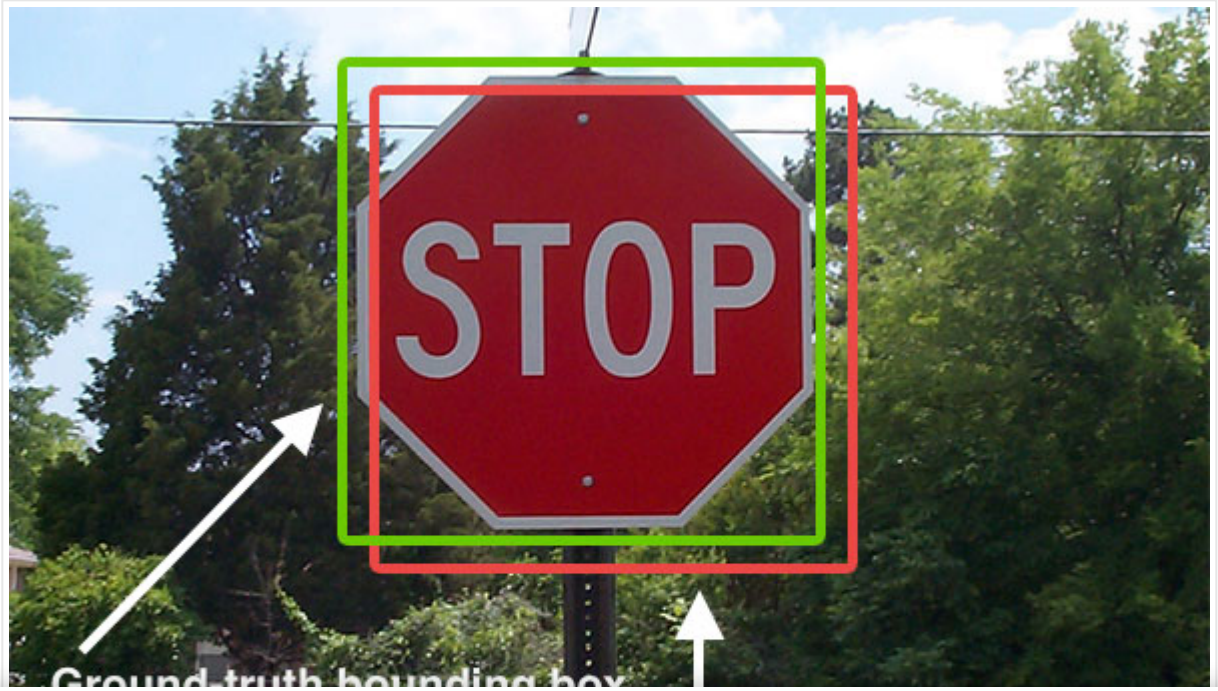
pyimagesearch
be awesome at building image search engines

Intersection over Union (IoU) for object detection

by **Adrian Rosebrock** on November 7, 2016 in **Machine Learning**, **Object Detection**, **Tutorials**



His professor mentioned that he should use the ***Intersection over Union (IoU)*** method for evaluation, but Jason's not sure how to implement it.

I helped Jason out over email by:

1. Describing *what* Intersection over Union is.
2. Explaining *why* we use Intersection over Union to evaluate object detectors.
3. Providing him with some *example Python code from my own personal library* to perform Intersection over Union on bounding boxes.

My email really helped Jason finish getting his final year project together and I'm sure he's going to pass with flying colors.

help you as well.

To learn how to evaluate your own custom object detectors using the Intersection over Union evaluation metric, just keep reading.

Looking for the source code to this post?
Jump right to the downloads section.

Intersection over Union for object detection

In the remainder of this blog post I'll explain *what* the Intersection over Union evaluation metric is and *why* we use it.

I'll also provide a Python implementation of Intersection over Union that you can use when evaluating your own custom object detectors.

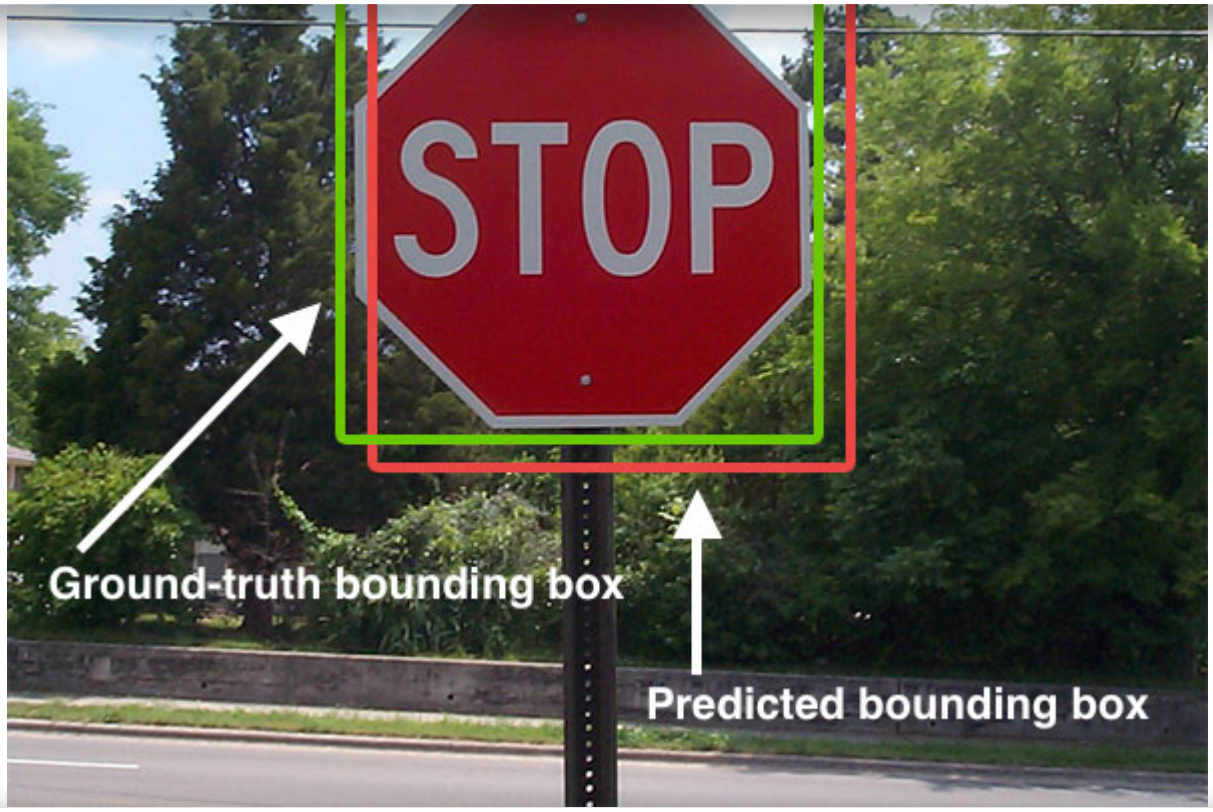
Finally, we'll look at some *actual results* of applying the Intersection over Union evaluation metric to a set of *ground-truth* and *predicted* bounding boxes.

More formally, in order to apply Intersection over Union to evaluate an (arbitrary) object detector we need:

1. The *ground-truth bounding boxes* (i.e., the hand labeled bounding boxes from the testing set that specify *where* in the image our object is).
2. The *predicted bounding boxes* from our model.

As long as we have these two sets of bounding boxes we can apply Intersection over Union.

Below I have included a visual example of a ground-truth bounding box versus a predicted bounding box:



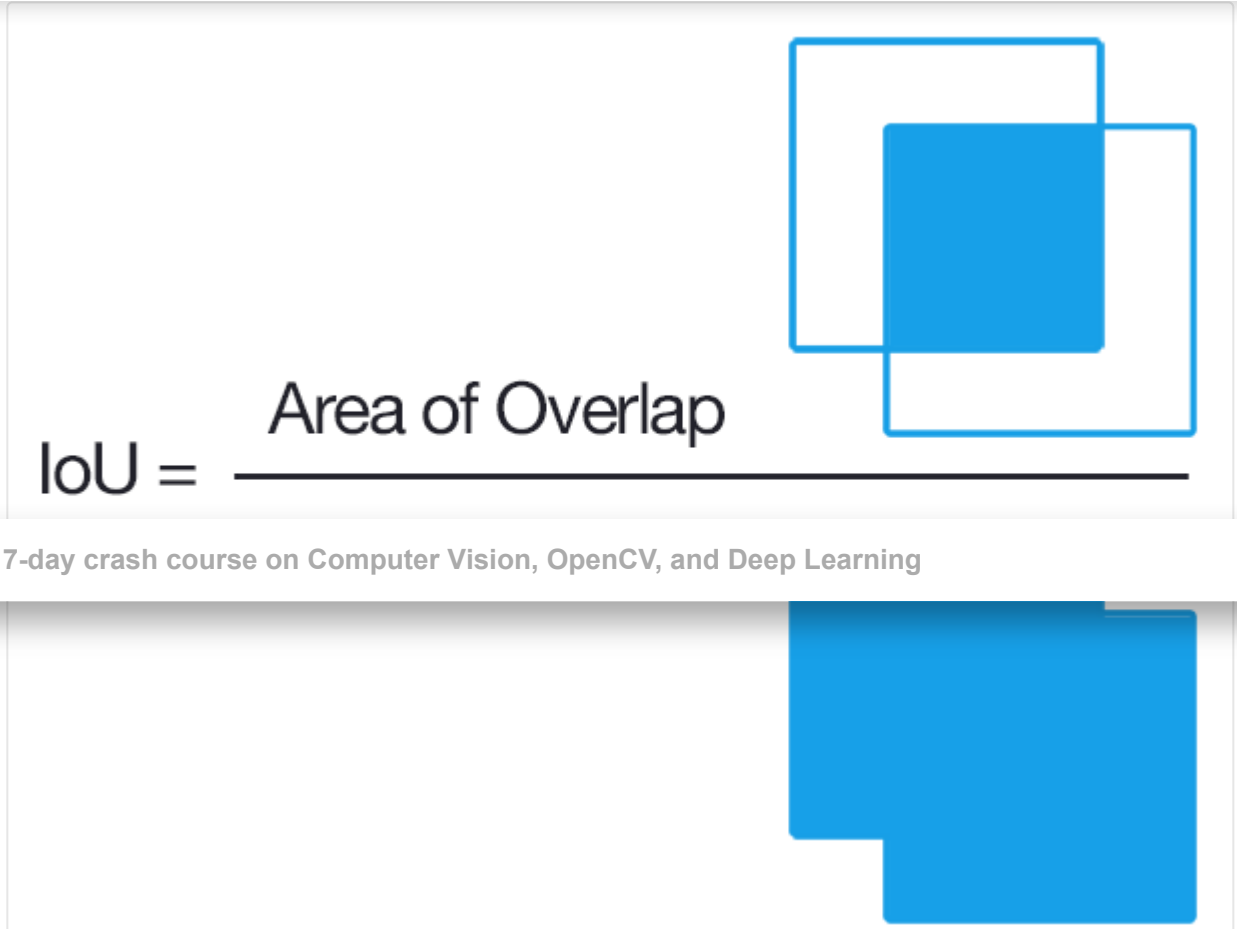


Figure 2: Computing the Intersection of Union is as simple as dividing the area of overlap between the bounding boxes by the area of union (thank you to the excellent Pittsburg HW4 assignment for the inspiration for this figure).

Examining this equation you can see that Intersection over Union is simply a ratio.

In the numerator we compute the ***area of overlap*** between the *predicted* bounding box and the *ground-truth* bounding box.

The denominator is the ***area of union***, or more simply, the area encompassed by *both* the predicted bounding box and the ground-truth bounding box.

You may also have a *validation set* used to tune the hyperparameters of your model.

Both the training and testing set will consist of:

1. The actual images themselves.
2. The *bounding boxes* associated with the object(s) in the image. The bounding boxes are simply the *(x, y)*-coordinates of the object in the image.

The bounding boxes for the training and testing sets are *hand labeled* and hence why we call them the "ground-truth".

Your goal is to take the training images + bounding boxes, construct an object detector, and then evaluate its performance on the testing set.

Why do we use Intersection over Union?

If you have performed any previous machine learning in your career, specifically classification, you'll likely be used to *predicting class labels* where your model outputs a single label that is either *correct* or *incorrect.*

This type of binary classification makes computing accuracy straightforward; however, for object detection it's not so simple.

In all reality, it's *extremely unlikely* that the *(x, y)*-coordinates of our predicted bounding box are going to ***exactly match*** the *(x, y)*-coordinates of the ground-truth bounding box.

Due to varying parameters of our model (image pyramid scale, sliding window size, feature extraction method, etc.), a complete and total match between predicted and ground-truth bounding boxes is simply unrealistic.

Because of this, we need to define an evaluation metric that *rewards* predicted bounding boxes for heavily overlapping with the ground-truth:

Figure 3: An example of computing Intersection over Unions for various bounding boxes.

In the above figure I have included examples of good and bad Intersection over Union scores.

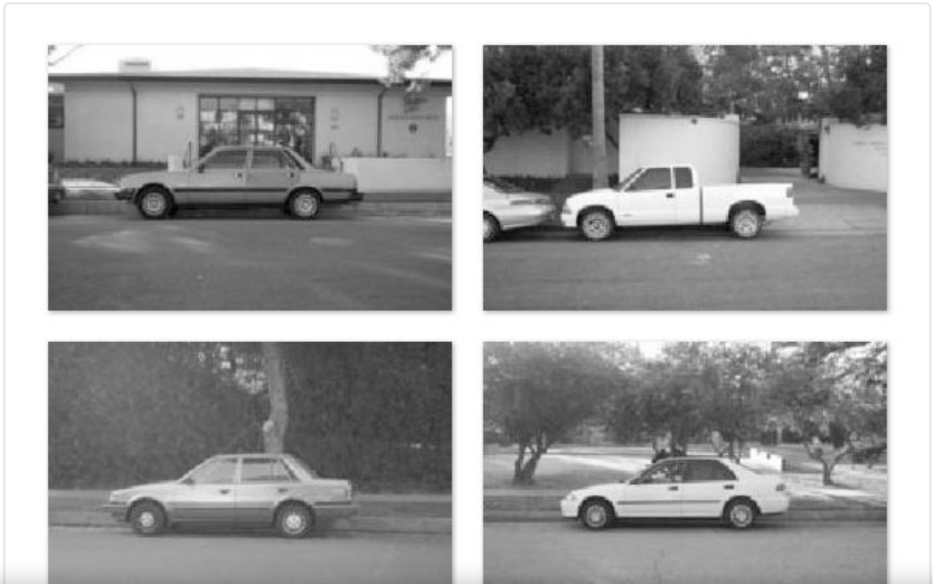
As you can see, predicted bounding boxes that heavily overlap with the ground-truth bounding boxes have higher scores than those with less overlap. This makes Intersection over Union an excellent metric for evaluating custom object detectors.

We aren't concerned with an *exact* match of *(x, y)*-coordinates, but we do want to ensure that our predicted bounding boxes match as closely as possible — Intersection over Union is able to take this into account.

Implementing Intersection over Union in Python

Now that we understand what intersection over Union is and why we use it to evaluate object detection models, let's go ahead and implement it in Python.

Before we get started writing any code though, I want to provide the five example images we will be working with:



Inside the ***PyImageSearch Gurus course*** I demonstrate how to train a custom object detector to detect the presence of cars in images like the ones above using the HOG + Linear SVM framework.

I have provided a visualization of the ground-truth bounding boxes (green) along with the predicted bounding boxes (red) from the custom object detector below:

Figure 5: Our goal is to evaluate the performance of our object detector by using Intersection of Union. Specifically, we want to measure the accuracy of the predicted bounding box (red) against the ground-truth (green).

Given these bounding boxes, our task is to define the Intersection over Union metric that can be used to evaluate how "good (or bad) our predictions are

- `image_path` : The path to our input image that resides on disk.
- `gt` : The ground-truth bounding box.
- `pred` : The predicted bounding box from our model.

As we'll see later in this example, I've already obtained the predicted bounding boxes from our five respective images and hardcoded them into this script to keep the example short and concise.

For a complete review of the HOG + Linear SVM object detection framework, please refer to this blog post. And if you're interested in learning more about training your own custom object detectors from scratch, be sure to check out the PyImageSearch Gurus course.

Let's go ahead and define the `bb_intersection_over_union` function, which as the name suggests, is responsible for computing the Intersection over Union between two bounding boxes:


```python
def bb_intersection_over_union(boxA, boxB):
    # determine the (x, y)-coordinates of the intersection rectangle
    xA = max(boxA[0], boxB[0])
    yA = max(boxA[1], boxB[1])
    xB = min(boxA[2], boxB[2])
    yB = min(boxA[3], boxB[3])

    # compute the area of intersection rectangle
    interArea = max(0, xB - xA + 1) * max(0, yB - yA + 1)

    # compute the area of both the prediction and ground-truth
    # rectangles
    boxAArea = (boxA[2] - boxA[0] + 1) * (boxA[3] - boxA[1] + 1)
    boxBArea = (boxB[2] - boxB[0] + 1) * (boxB[3] - boxB[1] + 1)

    # compute the intersection over union by taking the intersection
    # area and dividing it by the sum of prediction + ground-truth
    # areas - the interesection area
    iou = interArea / float(boxAArea + boxBArea - interArea)

    # return the intersection over union value
    return iou
```

This method requires two parameters: `boxA` and `boxB` , which are presumed to be our ground-truth and predicted bounding boxes (the actual *order* in which these parameters are supplied to

Finally, the Intersection over Union score is returned to the calling function on **Line 30**.

Now that our Intersection over Union method is finished, we need to define the ground-truth and predicted bounding box coordinates for our five example images:

```
Intersection over Union (IoU) for object detection                    Python
32 # define the list of example detections
33 examples = [
34     Detection("image_0002.jpg", [39, 63, 203, 112], [54, 66, 198, 114]),
35     Detection("image_0016.jpg", [49, 75, 203, 125], [42, 78, 186, 126]),
36     Detection("image_0075.jpg", [31, 69, 201, 125], [18, 63, 235, 135]),
37     Detection("image_0090.jpg", [50, 72, 197, 121], [54, 72, 198, 120]),
38     Detection("image_0120.jpg", [35, 51, 196, 110], [36, 60, 180, 108])]
```

As I mentioned above, in order to keep this example short(er) and concise, I have *manually obtained* the

For more information on how I trained this exact object detector, please refer to the PyImageSearch Gurus course.

We are now ready to evaluate our predictions:

```
Intersection over Union (IoU) for object detection                    Python
40 # loop over the example detections
41 for detection in examples:
42     # load the image
43     image = cv2.imread(detection.image_path)
44
45     # draw the ground-truth bounding box along with the predicted
46     # bounding box
47     cv2.rectangle(image, tuple(detection.gt[:2]),
48         tuple(detection.gt[2:]), (0, 255, 0), 2)
49     cv2.rectangle(image, tuple(detection.pred[:2]),
50         tuple(detection.pred[2:]), (0, 0, 255), 2)
51
52     # compute the intersection over union and display it
53     iou = bb_intersection_over_union(detection.gt, detection.pred)
54     cv2.putText(image, "IoU: {:.4f}".format(iou), (10, 30),
55         cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
56     print("{}: {:.4f}".format(detection.image_path, iou))
```

We then write the Intersection over Union value on the `image` itself followed by our console as well.

Finally, the output image is displayed to our screen on **Lines 59 and 60**.

Comparing predicted detections to the ground-truth with Intersection over Union

To see the Intersection over Union metric in action, make sure you have downloaded the source code + example images to this blog post by using the ***"Downloads"*** section found at the bottom of this tutorial.

After unzipping the archive, execute the following command:

Our first example image has an Intersection over Union score of *0.7980*, indicating that there is significant overlap between the two bounding boxes:

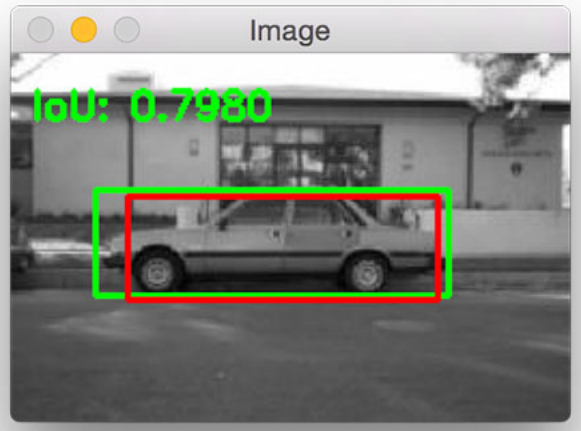


Figure 7: A slightly better Intersection over Union score.

Notice how the ground-truth bounding box (green) is wider than the predicted bounding box (red). This is because our object detector is defined using the HOG + Linear SVM framework which requires us to specify a fixed size sliding window (not to mention, an image pyramid scale and the HOG parameters themselves).

Ground-truth bounding boxes will naturally have a slightly different aspect ratio than the predicted bounding boxes, but that's okay provided that the Intersection over Union score is > 0.5 — as we can see, this still a great prediction.

The next example demonstrates a slightly "less good" prediction where our predicted bounding box is much less "tight" than the ground-truth bounding box:

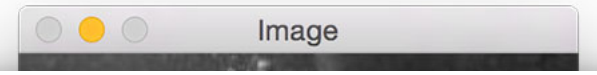


Figure 8: Deriving the Intersection of Union evaluation metric for object detection.

The reason for this is because our HOG + Linear SVM detector likely couldn't "find" the car in the lower layers of the image pyramid and instead fired near the top of the pyramid where the image is much smaller.

The following example is an *extremely good* detection with an Intersection over Union score of *0.9472*:

Figure 9: Measuring object detection performance using Intersection over Union.

Notice how the predicted bounding box nearly perfectly overlaps with the ground-truth bounding box.

Here is one final example of computing Intersection over Union:

Figure 10: Intersection over Union for evaluating object detection algorithms.

Want to train your own custom object detectors?

If you enjoyed this tutorial and want to learn more about training your own custom object detectors, you'll *definitely* want to take a look at the ***PyImageSearch Gurus course*** — the most *complete, comprehensive* computer vision course online today.

Inside the course, you'll find over **168 lessons** covering **2,161+ pages of content** on *Object Detection, Image Classification, Convolutional Neural Networks,* ***and much more.***

To learn more about the PyImageSearch Gurus course (and grab your ***FREE sample lessons + course syllabus***), just click the button below:

Click here to learn more about PyImageSearch Gurus!

Finally, before you go, be sure to enter your email address in the form below to be notified when future PyImageSearch blog posts are published — you won't want to miss them!

Downloads:

If you would like to download the code and images used in this post, please enter your email address in the form below. Not only will you get a .zip of the code, I'll also send you a **FREE 17-page Resource Guide on Computer Vision, OpenCV, and Deep Learning.** Inside you'll find my hand-picked tutorials, books, courses, and libraries to help you master CV and DL! Sound good? If so, enter your email address and I'll send you the code immediately!

wajih November 7, 2016 at 10:33 am #

REPLY

I was translating a code, was wondering of IoU, and now really I OWE YOU ONE 🙂 Thanks for the explanation. Helped me finish the translation in a breeze 🙂

Adrian Rosebrock November 7, 2016 at 2:42 pm #

REPLY

Awesome, I'm happy I could help Wajih! 🙂

Ha, I love the play of words… I o U 1 XD

Walid Ahmed November 7, 2016 at 11:23 am #

REPLY

Thanks a lot.

This was really in time for me
however, I am still clueless how to build a classifier for object detection.
I already built my classifier for object classification using CNN.
Any advice?

Adrian Rosebrock November 7, 2016 at 2:42 pm #

REPLY

Hey Walid — I would suggest starting by reading about the HOG + Linear SVM detector, a classic method used for object detection. I also demonstrate how to implement this system inside the PyImageSearch Gurus course. As for using a CNN for object detection, I will be covering that in my

Most of my work focuses on object detection rather than pixel-wise segmentations of an image but I'll certainly consider it for the future.

jsky November 7, 2016 at 10:43 pm #

REPLY

I implemented this in my car detection framework for my FYP too.
Its called the Jaccard Index, and is a standard measure for evaluating such record retrieval.
https://en.wikipedia.org/wiki/Jaccard_index

thank you for the tutorial

Adrian Rosebrock November 10, 2016 at 9:40 am #

REPLY

No problem, happy I can help!

Aamer November 12, 2016 at 2:33 am #

REPLY

what if our hog+svm model predicts multiple bounding boxes.

In that case, will we iterate over all such predicted bounding boxes and see for the one which gets the max value for the Intersection/Union ratio ?

Adrian Rosebrock November 14, 2016 at 12:12 pm #

REPLY

Miej November 28, 2016 at 11:14 am #

REPLY

that said, it's still quite handy. thanks!

auro November 28, 2016 at 2:34 pm #

REPLY

Do we need to consider the case where the two boxes to not intersect at all?
Look up the MATLAB code at https://github.com/rbgirshick/voc-dpm/blob/master/utils/boxoverlap.m

Adrian Rosebrock November 28, 2016 at 2:40 pm #

REPLY

Good point, thank you for pointing this out Auro.

Rimphy Darmanegara December 8, 2016 at 11:55 pm #

REPLY

So, in case of negative result just return zero?

Thank you for the code.

Adrian Rosebrock December 10, 2016 at 7:14 am #

REPLY

Correct. In case of negative (non-overlapping) objects the return value would be zero.

Anivini December 21, 2016 at 4:34 am #

REPLY

This is helpful knowledge. You have made me understand about the method IoU used in fast rcnn. Thanks you are one of the best people in explaining concepts easily

Adrian Rosebrock December 23, 2016 at 10:52 am #

REPLY

Thank you for the kind words Jere 🙂

sabrine December 29, 2016 at 8:54 am #

REPLY

please help me.

Adrian Rosebrock December 31, 2016 at 1:26 pm #

REPLY

If you have the predictions from your MATLAB detector you can either (1) write them to file and use my Python code to read them and compare them to the ground-truth or (2) implement this function in MATLAB.

sabrine January 1, 2017 at 6:04 am #

OK thanks
I will try to implement on matlab
Because I do not know how to use python

goingmyway January 15, 2017 at 4:56 am #

REPLY

Awesome post. A clear explanation of IoU.

Adrian Rosebrock January 15, 2017 at 12:03 pm #

REPLY

Thank you, I'm happy to hear you enjoyed it! 🙂

Hey I would like to know how to compute the repeatability factor ,corresponding count and recall and precision to evaluate feature detector in python

I would like to know is there a function in python similar to the one in C++ if not then how do I proceed

Paarijaat Aditya March 7, 2017 at 10:06 am #

REPLY

Very helpful post! Thanks! A small correction, line 17 should be:

interArea = max(0, xB – xA + 1) * max(0, yB – yA + 1)

This is to avoid those corner cases where the rectangles are not overlapping but the intersection area still computes to be greater than 0. This happens when both the brackets in the original line 17 are negative. For e.g. boxA is on the upper right of the picture and boxB is somewhere on the lower left of the picture, without any overlap and with significant separation between them.

Adrian Rosebrock March 8, 2017 at 1:07 pm #

REPLY

Adrian Rosebrock April 8, 2017 at 1:02 pm #

REPLY

Adding one simply prevents the numerator from being zero.

Johannes April 7, 2017 at 9:52 am #

REPLY

Thanks, helped me out understanding the YOLO9000 paper.

Fantastic, I'm glad to hear it Johannes!

Mohammad April 9, 2017 at 2:35 pm #

REPLY

Hey Adrian,
I tested your code but I think it is a little buggy:
Assume that we have two below bounding boxes: (the structure of bounding boxes are (x1,y1,x2,y2), it is just a tuple in python. (x1,y1) denotes to the top-left coordinate and (x2,y2) denotes to the bottom-right coordinate.)

boxA = (142,208,158,346)
boxB = (243,203,348,279)

Based on these BBoxes, the IoU should be zero. Because there is no intersection between them. But your code give a negative value! By the way, I should say that my origin is up-left corner of the image.
So how can I solve this problem?

@adrian That doesn't look enough to resolve the issue. `interArea = (xB - xA + 1) * (yB - yA + 1)` could be positive when two terms are both negative. It should return 0 if either `(xB - xA + 1)` or `(yB - yA + 1)` is equal or less than 0.

Filippo November 20, 2017 at 6:14 am #

REPLY

Please, fix this issue, IOU should be 0 when interArea is the product of two negative terms. Not that you owe anyone anything, but this is the first result on google when searching intersection over union, it would be great not to have to scroll down to the comments to find out the code is buggy. Paarijaat Aditya solution works pretty fine and handles both corner cases of non overlapping boxes

Islam Saad April 27, 2017 at 4:53 am #

REPLY

Many thanks Adrian for the great topic You.
I wanna ask if I have dataset groundtruth as contour shows all object outline (not rectangle box shape). Then, is there another method dealing with free shape bounding contours? In order to compute Lou.

Adrian Rosebrock April 28, 2017 at 9:32 am #

REPLY

Intersection over Union assumes bounding boxes. You can either convert your contour points to a bounding box and compute IoU or you can simply count the number of pixels in your mask that overlap with the detection.

Javier June 20, 2017 at 9:12 pm #

REPLY

MMD October 21, 2017 at 5:23 am #

REPLY

Hi Adrian Please am interested in your project but am not a developer.
I will like you to develop a project for me the intended purpose is mainly to Classify Vehicle base on Length, Height and Axle count.
Please any one interested in this commercial project is welcome.
Thanks

Adrian Rosebrock October 22, 2017 at 8:34 am #

REPLY

who can help you build your project.

K M Ibrahim Khalilullah June 29, 2017 at 2:04 am #

REPLY

Thanks for this tutorial

K M Ibrahim Khalilullah July 23, 2017 at 12:06 am #

REPLY

How can I calculate precision-recall from IoU?

Walid Ahmed August 16, 2017 at 2:37 pm #

REPLY

Thanks
But I found something strange

Adrian Rosebrock September 5, 2017 at 9:31 am #

REPLY

Thanks Oyesh! 🙂

hema January 13, 2018 at 6:53 am #

REPLY

Hi Adrian Rosebrock,
I am trying a HOG descriptor with SVM classifier to detect objects.And after creating annotation xml file if i use it for training it gives me the following error: “An impossible set of object boxes was given for training.All the boxes need to have a similar aspect ratio and also not be smaller than 400 pixels in

Adrian Rosebrock January 15, 2018 at 9:24 am #

REPLY

I assume you are using dlib to train your own custom object detector? The HOG + Linear SVM implementation in dlib requires your bounding boxes to have a similar aspect ratio (i.e., similar width and height of the ROI). It sounds like your objects do not have this or you have an invalid bounding box in your annotations file.

Mohamed Judi January 20, 2018 at 9:14 am #

REPLY

Hi Adrian, thank you so much for this excellent blog. It explains IoU very well. It is a gift to find someone, not only knows his stuff but also knows how to explain it to people in simple terms.

I’m working on the Kaggle’s 2018 Data Science Bowl competition with very little knowledge of deep learning, let alone Biomedical Image Segmentation and U-Nets 🙂 However, I’m taking this challenge to learn. Winning is secondary at this stage of my career.

Adrian Rosebrock January 22, 2018 at 6:34 pm #

REPLY

There are two components to need to consider here (as is true with object detection): precision and recall. You first need to detect the *correct* object. In the case of object detection and semantic segmentation, this is your recall. For example, if you detect a “cat” but the actual label is a dog, then your recall score goes down.

To measure the precision (accuracy) of the detection/segmentation we can use IoU. Exactly how IoU is used for segmentation depends on the challenge so you should consult the Kaggle documentation and/or evaluation scripts but typically it’s the ratio of ground-truth mask and the predicted mask.

Thank you, Adrian!

Qaisar Tanvir January 23, 2018 at 3:21 am #

REPLY

How can we do this with polygons? where bounding box may be a little rotated (RBOX). i dont have the angle of rotation. So my output bounding box cannot be drawn with top-left and bottom-right. it has four points. Any help or suggestion will be highly appreciated guys

Adrian Rosebrock January 23, 2018 at 2:00 pm #

REPLY

Why not convert your rotated bounding box to a “normal” bounding box and then compute IoU on the non-rotated version?

REPLY

Indeed, you are right Glen. I will get the code updated.

Johannes May 18, 2018 at 9:56 am #

REPLY

Hi, I am not 100% sure, but I think that your code tends to overestimate the IoU.

Given two boxes [[0,0], [10, 10]] and [[1,1], [11, 11]]. The area of intersection should be 81. However, with your function it computes 100. The same problem occurs with the box size which are obviously 100 but are computed as 121. Your code gives an IoU of 0.7, while the analytical solution should be 0.68…

Maybe you could clarify, why you are always adding +1 to the coordinates. This somehow seems to deviate from the standard. Used in many other implementations.

Adrian Rosebrock May 22, 2018 at 6:43 am #

REPLY

The “+1” here is used to prevent any division by zero errors. That said, I will reinvestigate this implementation in the near future.

ptyshevs July 10, 2018 at 1:38 am #

REPLY

Hi, I’ve decided to check computations of IoU by hand and it seems that “+1” in your code is responsible for the incorrect result.

Let bboxA = [0, 0, 2, 2], bboxB = [1, 1, 3, 3], then Union(bboxA, bboxB) = 7, Intersection(bboxA, bboxB) = 1, yielding IoU = 1/7 = 0.1428…
Your version will give 0.2857…

I suggest the following snippet:

interArea = abs((xB – xA) * (yB – yA))
if interArea == 0:

Name (required)

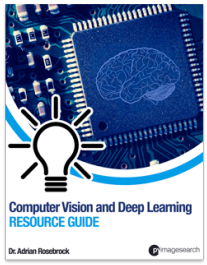
Email (will not be published) (required)

Free 17-day crash course on Computer Vision, OpenCV, and Deep Learning

SUBMIT COMMENT

Search...

Resource Guide (it's totally free).



Get your **FREE 17 page Computer Vision, OpenCV, and Deep Learning Resource Guide PDF.** Inside you'll find my hand-picked tutorials, books, courses, and libraries to help you master CV and DL.

Download for Free!

Deep Learning for Computer Vision with Python Book — OUT NOW!

Free 17-day crash course on Computer Vision, OpenCV, and Deep Learning

Interested in computer vision, OpenCV, and deep learning, but don't know where to start? Let me help. I've created a *free*, 17-day crash course that is hand-tailored to give you the best possible introduction to computer vision and deep learning. Sound good? Enter your email below to get started.

Email Address

START MY EMAIL COURSE

The PyImageSearch Gurus course is *now enrolling!* Inside the course you'll learn how to perform:

- Automatic License Plate Recognition (ANPR)
- Face Recognition
- *and much more!*

Click the button below to learn more about the course, take a tour, and get 10 (FREE) sample lessons.

TAKE A TOUR & GET 10 (FREE) LESSONS

Hello! I'm Adrian Rosebrock.

I'm an entrepreneur and Ph.D who has launched two successful image search engines, ID My Pill and Chic Engine. I'm here to share my tips, tricks, and hacks I've learned along the way.

Learn computer vision in a single weekend.

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3 APRIL 18, 2016
Install OpenCV and Python on your Raspberry Pi 2 and B+ FEBRUARY 23, 2015
Raspbian Stretch: Install OpenCV 3 + Python on your Raspberry Pi SEPTEMBER 4, 2017
Home surveillance and motion detection with the Raspberry Pi, Python, OpenCV, and Dropbox JUNE 1, 2015
Ubuntu 16.04: How to install OpenCV OCTOBER 24, 2016
How to install OpenCV 3 on Raspbian Jessie OCTOBER 26, 2015
Basic motion detection and tracking with Python and OpenCV MAY 25, 2015