# 2.8: Non-maxima suppression


(https://gurus.pyimagesearch.com/wp-content/uploads/2015/06/nms_image_0004.jpg)

There's an inescapable issue you must handle when building object detection systems — *overlapping bounding boxes*. It's going to happen; there's no way around it. In fact, it's actually a good sign that your object detector is firing properly, so I wouldn't even call it a true "issue" at all. It would be far worse if your detector (1) reported a false positive (i.e., detected an object where there isn't one) or (2) failed to detect an object altogether.

To handle the removal of overlapping bounding boxes (that refer to the same object), we can use either non-maxima suppression (NMS) or the Mean-Shift algorithm (http://en.wikipedia.org/wiki/Mean-shift). While Dalal and Triggs (https://gurus.pyimagesearch.com/wp-

content/uploads/2015/05/dalal_2005.pdf) prefer using Mean-Shift, I find that Mean-Shift gives sub-par results, and in nearly *all* cases, non-maxima suppression should be utilized instead.

In the remainder of this blog post, I'll show you how to implement and use non-maxima suppression for object detection. This implementation will allow us to resolve the multiple, overlapping bounding box problem.

# Objectives:

In this lesson, we will:

- Implement non-maxima suppression to reduce overlapping bounding boxes.

# Non-maxima suppression

The goal of non-maxima suppression is to compute the ratio of overlap between bounding boxes, then suppress (i.e., remove) bounding boxes that have significant overlap.

Let's go ahead and implement non-maxima suppression. This particular implementation is a fusion of the Felzenszwalb et al. (http://www.pyimagesearch.com/2014/11/17/non-maximum-suppression-object-detection-python/) and the Malisiewicz et al. method (http://www.pyimagesearch.com/2015/02/16/faster-non-maximum-suppression-python/), both of which were originally implemented in MATLAB, and that I later translated to Python:

nms.py Python

```
1  # import the necessary packages
2  import numpy as np
3
4  def non_max_suppression(boxes, probs, overlapThresh):
5      # if there are no boxes, return an empty list
6      if len(boxes) == 0:
7          return []
8
9      # if the bounding boxes are integers, convert them to floats -- this is important since
10     # we'll be doing a bunch of divisions
11     if boxes.dtype.kind == "i":
12         boxes = boxes.astype("float")
13
14     # initialize the list of picked indexes
15     pick = []
16
17     # grab the coordinates of the bounding boxes
18     x1 = boxes[:, 0]
19     y1 = boxes[:, 1]
20     x2 = boxes[:, 2]
21     y2 = boxes[:, 3]
22
23     # compute the area of the bounding boxes and sort the bounding boxes by their associated
24     # probabilities
25     area = (x2 - x1 + 1) * (y2 - y1 + 1)
26     idxs = np.argsort(probs)
```

We start off on **Line 2** by importing a single package, NumPy, which we'll utilize for numerical processing.

From there, we define our `non_max_suppression` on **Line 4**. This function accepts three arguments, the first being our set of bounding boxes in the form of *(startX, startY, endX, endY)*. The second argument required is the *probabilities* associated with each of the `boxes` . Finally, we need to supply an *overlap threshold*. I'll discuss the overlap threshold a little later in this lesson.

**Lines 6 and 7** make a check on the bounding box. If there are no bounding boxes in the list, we simply return an empty list back to the caller.

**Line 11** makes another check, this time to see if the `boxes` array is of the "integer" data type. Since we will be doing a bunch of divisions, we'll convert the bounding box coordinates to floats on **Line 12**.

From there, we initialize our list of "picked" bounding boxes (i.e, the bounding boxes that we would like to keep, discarding the rest) on **Line 15**.

Next up, we'll go ahead and unpack the *(x, y)*-coordinates for each corner of the bounding box on **Lines 18-21** — this is done using simple NumPy array slicing.

We then compute the *area* of each bounding box on **Line 25** using our sliced *(x, y)*-coordinates.

Be sure to pay attention to **Line 26**. We apply the `np.argsort` function to grab all indexes of the ***sorted*** probabilities associated with the bounding boxes. It's important we sort on these probabilities as we're more confident in bounding boxes with larger probabilities, and thus want to retain them over bounding boxes with smaller probabilities.

Now, let's go into the meat of the non-maxima suppression function:

nms.py — Python

```
28      # keep looping while some indexes still remain in the indexes list
29      while len(idxs) > 0:
30          # grab the last index in the indexes list and add the index value to the list of
31          # picked indexes
32          last = len(idxs) - 1
33          i = idxs[last]
34          pick.append(i)
```

We start looping over our indexes on **Line 29**, where we will keep looping until we run out of indexes to examine.

From there, we'll grab the length of the `idx` list on **Line 32**, grab the value of the last entry in the `idx` list on **Line 33**, and append the index `i` to the list of bounding boxes we want to keep on **Line 34**.

Now it's time to compute the overlap ratios and determine which bounding boxes we can ignore:

nms.py — Python

```
36          # find the largest (x, y) coordinates for the start of the bounding box and the
37          # smallest (x, y) coordinates for the end of the bounding box
38          xx1 = np.maximum(x1[i], x1[idxs[:last]])
39          yy1 = np.maximum(y1[i], y1[idxs[:last]])
40          xx2 = np.minimum(x2[i], x2[idxs[:last]])
41          yy2 = np.minimum(y2[i], y2[idxs[:last]])
42
43          # compute the width and height of the bounding box
44          w = np.maximum(0, xx2 - xx1 + 1)
45          h = np.maximum(0, yy2 - yy1 + 1)
46
47          # compute the ratio of overlap
48          overlap = (w * h) / area[idxs[:last]]
49
50          # delete all indexes from the index list that have overlap greater than the
51          # provided overlap threshold
52          idxs = np.delete(idxs, np.concatenate(([last],
53              np.where(overlap > overlapThresh)[0])))
54
55      # return only the bounding boxes that were picked
56      return boxes[pick].astype("int")
```

**Lines 38-41** use the vectorized `np.maximum` and `np.minimum` functions to find the *largest* (*x*, *y*)-coordinates for the start of the bounding box and the *smallest* (*x*, *y*)-coordinates for the end of the bounding box.

Using these coordinates, we can then compute the width and height of the bounding box on **Lines 44 and 45**.

The width and height then allow us to compute the ratio of overlap on **Line 48**.

Given the ratio of overlap, **Lines 52 and 53** delete all indexes from the index list that have an `overlap` greater than the `overlapThresh` provided as an argument to the `non_max_suppression` function.

Finally, a list of "picked" bounding boxes are returned to the calling function on **Line 56**.

Now that our `non_max_suppression` function is defined, let's created a new file named `test_model.py`. This script will have essentially the same contents as `test_model_no_nms.py`, only this time we are going to apply non-maxima suppression to help solve the overlapping bounding box problem:

test_model.py — Python

```
# import the necessary packages
from pyimagesearch.object_detection import non_max_suppression
from pyimagesearch.object_detection import ObjectDetector
from pyimagesearch.descriptors import HOG
from pyimagesearch.utils import Conf
import numpy as np
import imutils
import argparse
import pickle
import cv2

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-c", "--conf", required=True, help="path to the configuration file")
ap.add_argument("-i", "--image", required=True, help="path to the image to be classified")
args = vars(ap.parse_args())

# load the configuration file
conf = Conf(args["conf"])

# load the classifier, then initialize the Histogram of Oriented Gradients descriptor
# and the object detector
model = pickle.loads(open(conf["classifier_path"], "rb").read())
hog = HOG(orientations=conf["orientations"], pixelsPerCell=tuple(conf["pixels_per_cell"]),
    cellsPerBlock=tuple(conf["cells_per_block"]), normalize=conf["normalize"], block_norm="L1")
od = ObjectDetector(model, hog)
```

The first code block of `test_model.py` is essentially identical to the `test_model_no_nms.py` script from our previous lesson. The only difference here is that we are now importing our `non_max_suppression` function from the `object_detection` sub-module.

test_model.py Python

```
28 # load the image and convert it to grayscale
29 image = cv2.imread(args["image"])
30 image = imutils.resize(image, width=min(260, image.shape[1]))
31 gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
32
33 # detect objects in the image and apply non-maxima suppression to the bounding boxes
34 (boxes, probs) = od.detect(gray, conf["window_dim"], winStep=conf["window_step"],
35     pyramidScale=conf["pyramid_scale"], minProb=conf["min_probability"])
36 pick = non_max_suppression(np.array(boxes), probs, conf["overlap_thresh"])
37 orig = image.copy()
38
39 # loop over the original bounding boxes and draw them
40 for (startX, startY, endX, endY) in boxes:
41     cv2.rectangle(orig, (startX, startY), (endX, endY), (0, 0, 255), 2)
42
43 # loop over the allowed bounding boxes and draw them
44 for (startX, startY, endX, endY) in pick:
45     cv2.rectangle(image, (startX, startY), (endX, endY), (0, 255, 0), 2)
46
47 # show the output images
48 cv2.imshow("Original", orig)
49 cv2.imshow("Image", image)
50 cv2.waitKey(0)
```

The first half of this code block also looks very familiar. We start off by loading our image, resizing it, and loading our classifier, as well as instantiating our HOG descriptor and `ObjectDetector` class.

The big change comes on **Line 36**, where we pass in the bounding boxes and associated probabilities to the `non_max_suppression` function. Bounding boxes that overlap by more than *0.3* (defined inside our configuration file) are suppressed, where the bounding box with the larger probability is retained.

The `non_max_suppression` function returns the set of suppressed bounding boxes that we can use to report our *final* object detections.

**Lines 40 and 41** draw the *original*, *non-suppressed* bounding boxes while **Lines 44 and 45** draw the final bounding boxes after applying non-maxima suppression. Finally, **Lines 48-50** show our output images.

To see our updated script in action, just execute the following command:

test_model.py Shell

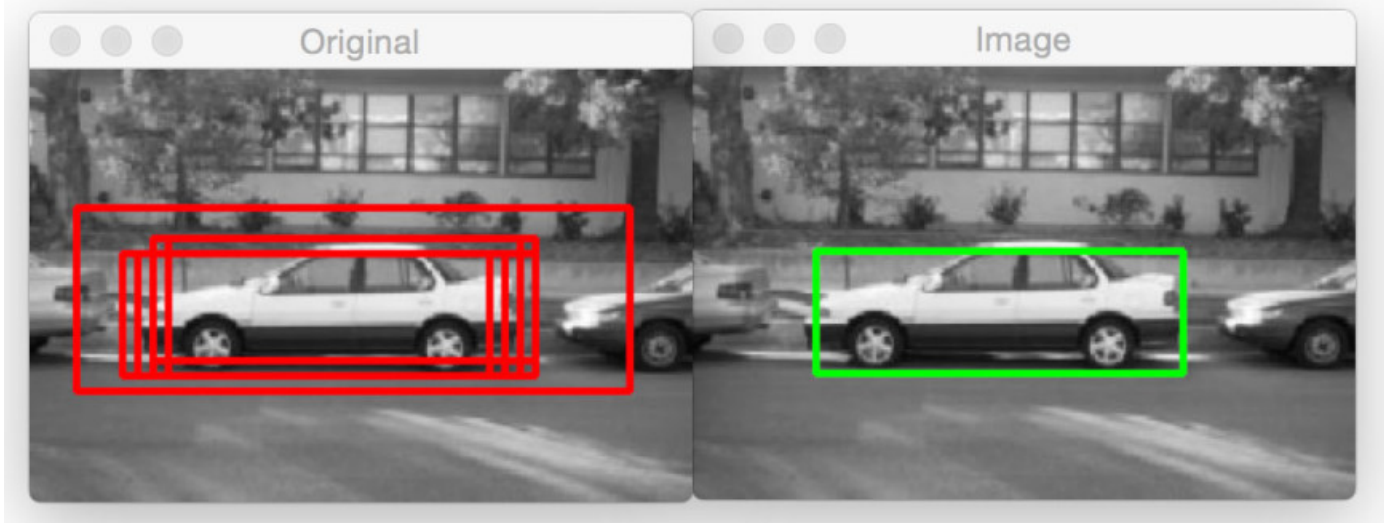
```
1 $ python test_model.py --conf conf/cars.json \
2     --image datasets/caltech101/101_ObjectCategories/car_side/image_0004.jpg
```


(https://gurus.pyimagesearch.com/wp-content/uploads/2015/06/nms_image_0004.jpg)

**FIGURE 1:** APPLYING NON-MAXIMA SUPPRESSION HAS REDUCED THE 11 BOUNDING BOXES TO ONLY A SINGLE BOUNDING BOX.

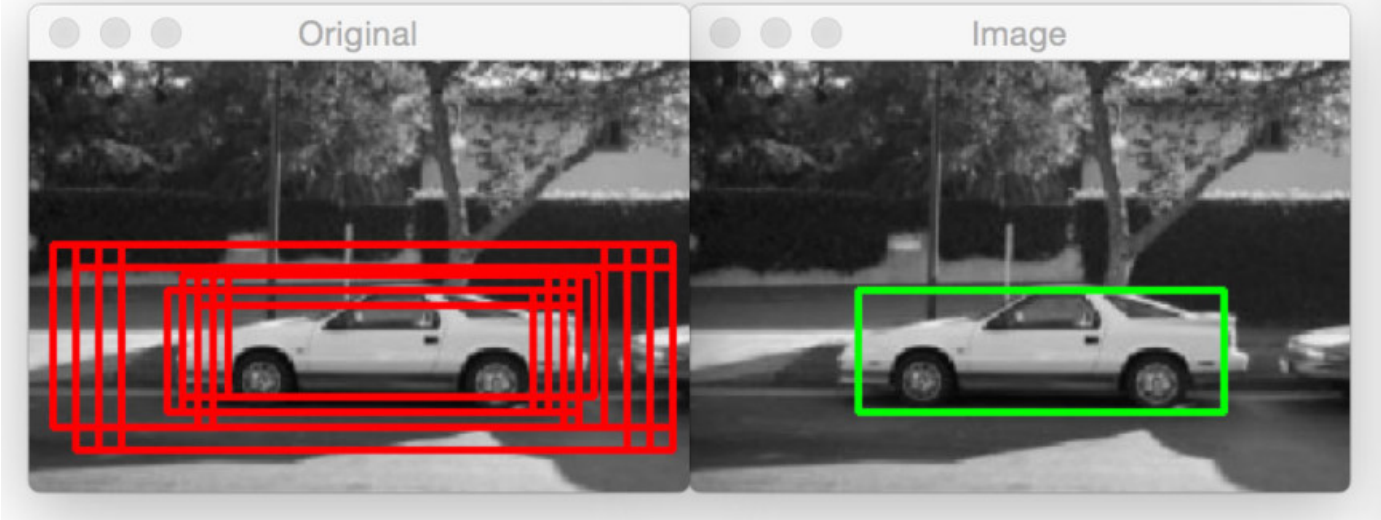
As we can see, we have reduced the 11 bounding boxes to only one bounding box due to the overlap.

Here are a few more examples of applying NMS:



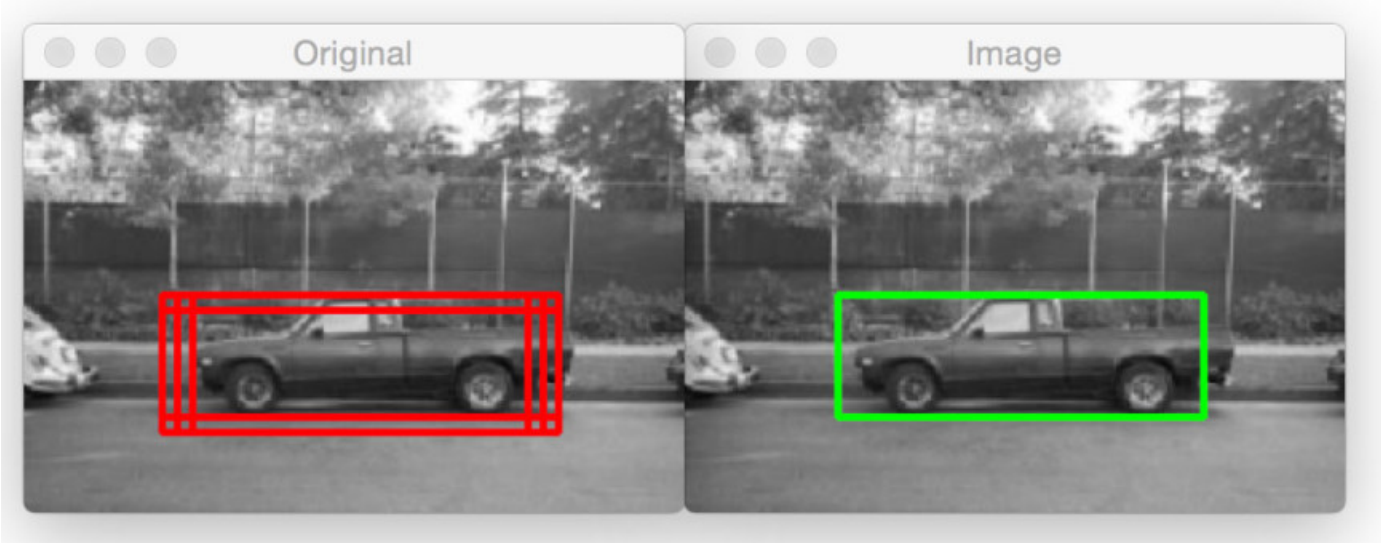

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/06/nms_image_0009.jpg)

**FIGURE 2:** NMS IS APPLIED TO REDUCE THE NUMBER OF OVERLAPPING BOUNDING BOXES.




(https://gurus.pyimagesearch.com/wp-content/uploads/2015/06/nms_image_0012.jpg)

**FIGURE 3:** EVEN THOUGH MULTIPLE, OVERLAPPING BOUNDING BOXES ARE DETECTED, NON-MAXIMA SUPPRESSION CAN QUICKLY SUPPRESS THEM INTO A SINGLE BOUNDING BOX.




(https://gurus.pyimagesearch.com/wp-content/uploads/2015/06/nms_image_0020.jpg)

**FIGURE 4:** NMS IS VERY FAST, MUCH FASTER AND MORE ACCURATE THAN THE MEAN-SHIFT METHOD PROPOSED BY DALAL AND TRIGGS.

Feedback

However, applying non-maxima suppression does not solve our problem of false-positive detection:

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/06/nms_false_positives.png)

**FIGURE 5:** NMS CAN BE APPLIED TO REDUCE OVERLAPPING BOUNDING BOXES; HOWEVER, IT DOES NOT RESOLVE THE ISSUE OF FALSE-POSITIVE DETECTIONS.

We will resolve this issue in our next lesson by applying *hard-negative mining* and *re-training* our classifier using the hard-negatives. This will result in a more accurate HOG + Linear SVM object detector that is less prone to misfires and false-positive detections.

# Summary

Having overlapping bounding boxes is a positive sign that our object detector is functioning properly. It indicates that our classifier is detecting regions surrounding the object as positive classifications. However, while this may be a positive sign, it's not exactly practical and ideal, especially if we need to *count* the number of objects in an image.

To resolve this problem, we implemented non-maxima suppression, a function that can be used to reduce overlapping bounding boxes to only a single bounding box, thus representing the true detection of the object. While Dalal and Triggs may prefer to use Mean-Shift, I find that non-maxima suppression ***consistently*** yields better results and should be used above Mean-Shift in nearly all situations.

However, non-maxima suppression cannot account for false-positive detections by our classifier. To remedy this issue, we need to apply *hard-negative mining*, a concept that we'll explore in our next lesson.

# Downloads:

Download the Code (https://gurus.pyimagesearch.com/protected/code/object_detector/nms.zip)

| Quizzes | Status |
| --- | --- |
| 1 | Non-Maxima Suppression Quiz (https://gurus.pyimagesearch.com/quizzes/non-maxima-suppression-quiz/) |

← Previous Lesson (https://gurus.pyimagesearch.com/lessons/the-initial-training-phase/) Next Lesson → (https://gurus.pyimagesearch.com/lessons/hard-negative-mining/)

## Course Progress

## Ready to continue the course?

Click the button below to **continue your journey to computer vision guru**.

I'm ready, let's go! (/pyimagesearch-gurus-course/)



## Resources & Links

- PyImageSearch Gurus Community (https://community.pyimagesearch.com/)
- PyImageSearch Virtual Machine (https://gurus.pyimagesearch.com/pyimagesearch-virtual-machine/)
- Setting up your own Python + OpenCV environment (https://gurus.pyimagesearch.com/setting-up-your-python-opencv-development-environment/)
- Course Syllabus & Content Release Schedule (https://gurus.pyimagesearch.com/course-syllabus-content-release-schedule/)
- Member Perks & Discounts (https://gurus.pyimagesearch.com/pyimagesearch-gurus-discounts-perks/)
- Your Achievements (https://gurus.pyimagesearch.com/achievements/)
- Official OpenCV documentation (http://docs.opencv.org/index.html)

## Your Account

- Account Info (https://gurus.pyimagesearch.com/account/)
- Support (https://gurus.pyimagesearch.com/contact/)
- Logout (https://gurus.pyimagesearch.com/wp-login.php?action=logout&redirect_to=https%3A%2F%2Fgurus.pyimagesearch.com%2F&_wpnonce=5736b21cae)