ROC for Feature Detection

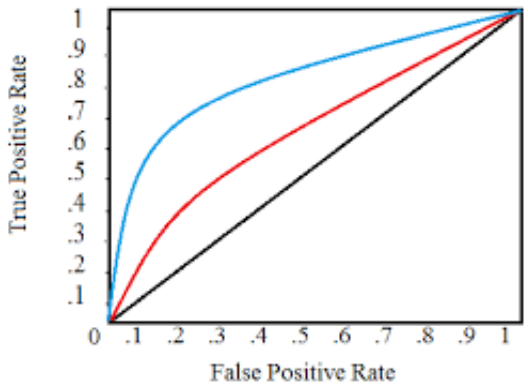
February 20, 2018

1 Receiver Operating Characteristic Curve for Feature Detection

1.1 William Koehrsen wjk68

The ROC curve is a standard tool for assessing the performance of a classifier. We can apply it to the feature detection model to determine how well it works and also explore ways to improve the performance.

ROC Curve



The ROC curve plots the **true positive rate (sensitivity)**:

$$\frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

versus the **false positive rate (1 - specificity)**:

$$\frac{\text{false positives}}{\text{true negatives} + \text{false positives}}$$

The true positives, false positives, true negatives, and false negatives will have to be determined by hand because the ground truth is what we as humans identify. To calculate the rates, we can construct a confusion matrix which shows the number of actual values and number of predicted values.

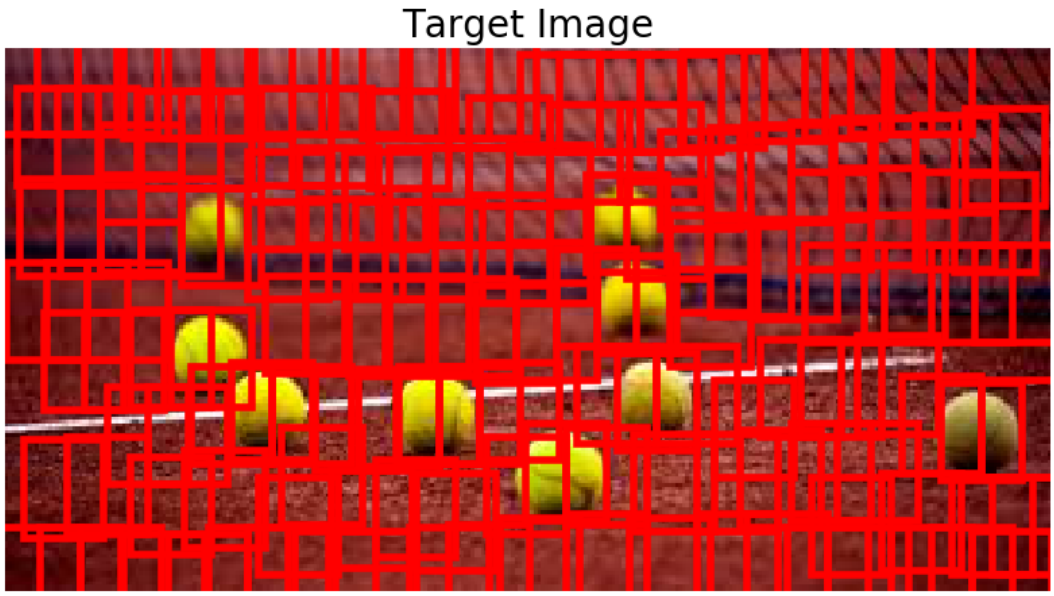
Confusion Matrix		
	Actual 1	**Actual 0**
Predicted 1	True Postive	False Positive
Predicted 0	False Negative	True Negative

In [1]:
```
import numpy as np

import matplotlib.pyplot as plt
import matplotlib

%matplotlib inline


from detector import model
```

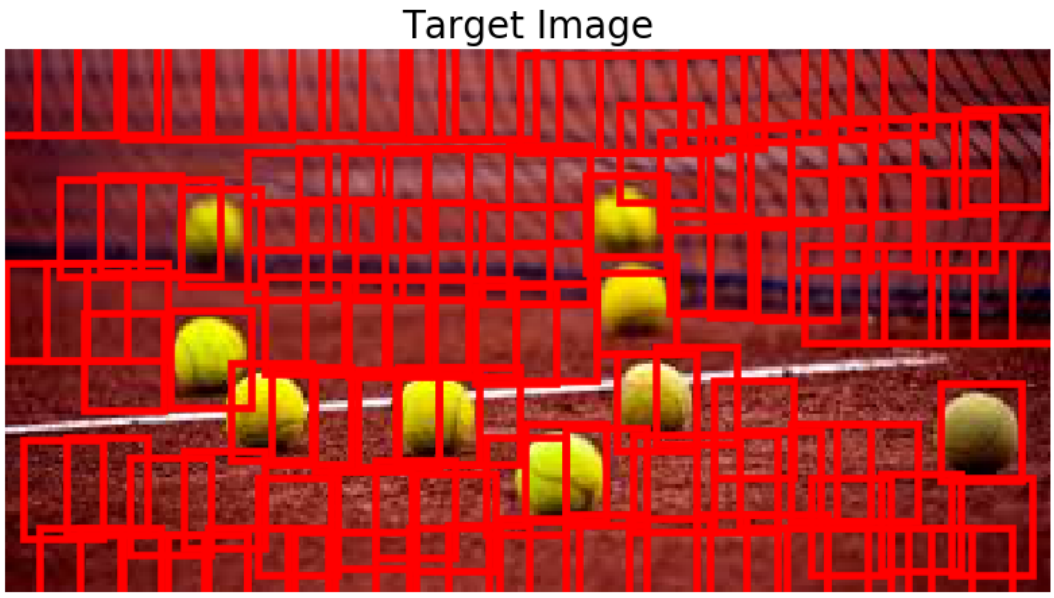
1.2 Construct Confusion Matrix

In [2]:
```
threshold_list = [0.5, 0.6, 0.7, 0.8, 0.9, 0.99]

for threshold in threshold_list:
    model(image_path='images/target_tennis.jpg', template_path='images/template_tennis
          threshold=threshold, analyze=True)
```

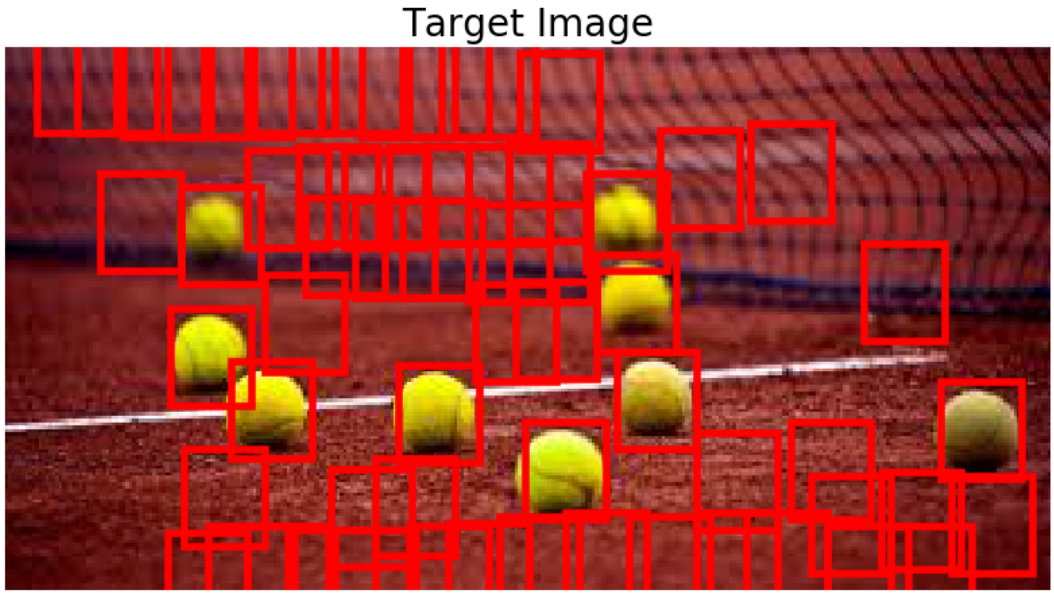
```
149 detections with threshold: 0.5
```



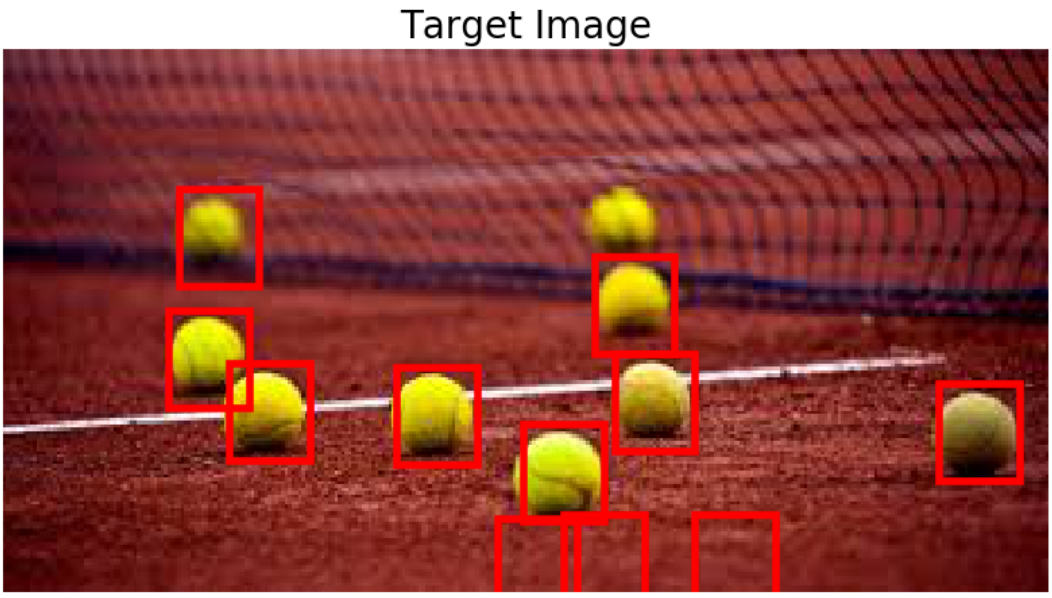

```
117 detections with threshold: 0.6
```



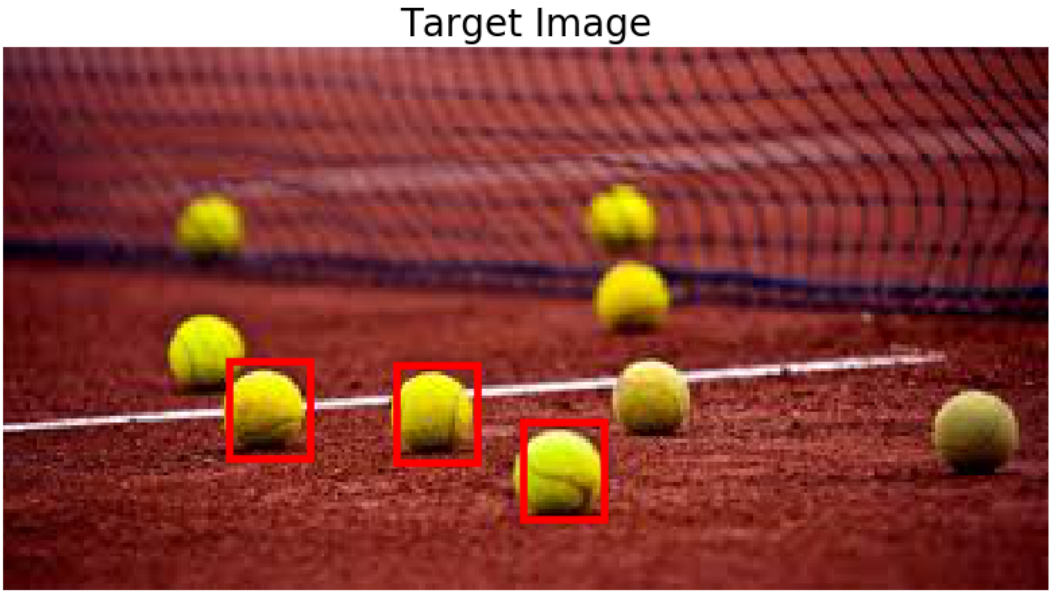

```
61 detections with threshold: 0.7
```



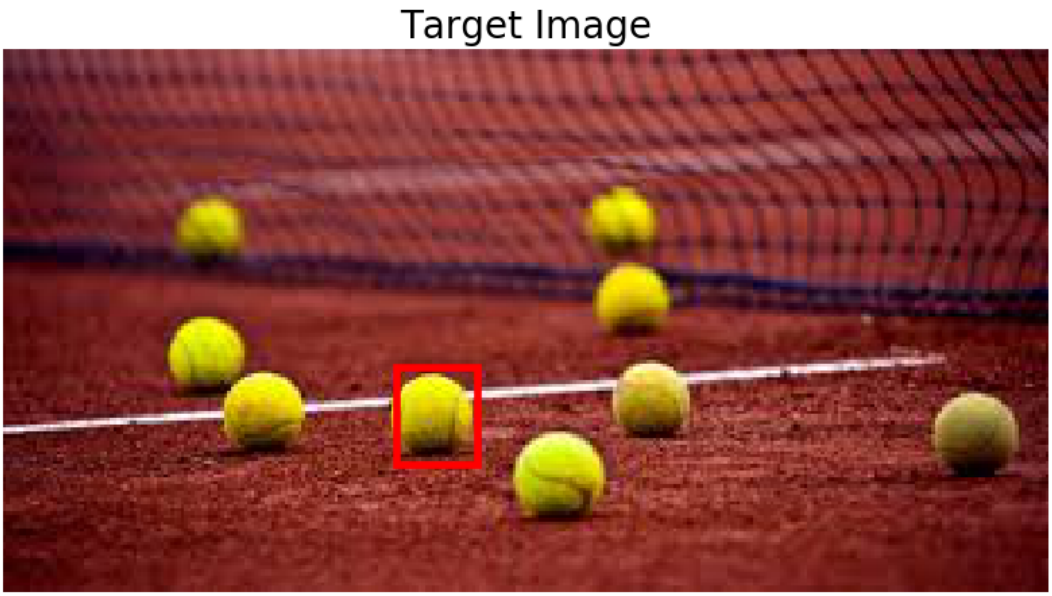

```
11 detections with threshold: 0.8
```




```
3 detections with threshold: 0.9
```




```
1 detections with threshold: 0.99
```



1.2.1 Ground Truth Values

The number of actual positives is easy: it is simply the number of features in the image, in this example 9 tennis balls. A perfect model would therefore identify all 9 tennis balls without classify any other points as matches

Defining acutal negatives is difficult in this case. I will say that actual negatives are every possible patch which does not contain the feature. As the convolution calculates the response of every single pixel, the number of possible patches is the pixel width times the pixel height of the image. The number of actual negatives is therefore the number of patches - the number of features in the image. For example, our image is 162 * 312 and there are 9 occurences of the feature in the image sothere are $(162 * 312) - 9 = 50535$ actual negatives in the image. These actual negatives are locations where the model should *not* identify a feature. Defining the number of actual negatives and the number of actual positives allows us to calculate the true negatives and false positives.

```
In [3]: image = plt.imread('images/target_tennis.jpg')
        actual_negatives = (image.shape[0] * image.shape[1]) - 9
        print('Number of actual negatives:', actual_negatives)
```

```
Number of actual negatives: 50535
```

1.2.2 First List of Results

Following is the true positives, false positives, true negatives, and false negatives for each of the thresholds. Using these numbers, we can calculate the

true positive rate:

$$\frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

for the y-axis
and the **false positive rate**:

$$\frac{\text{false positives}}{\text{true negatives} + \text{false positives}}$$

for the x-axis. This forms the ROC curve as a function of the threshold.

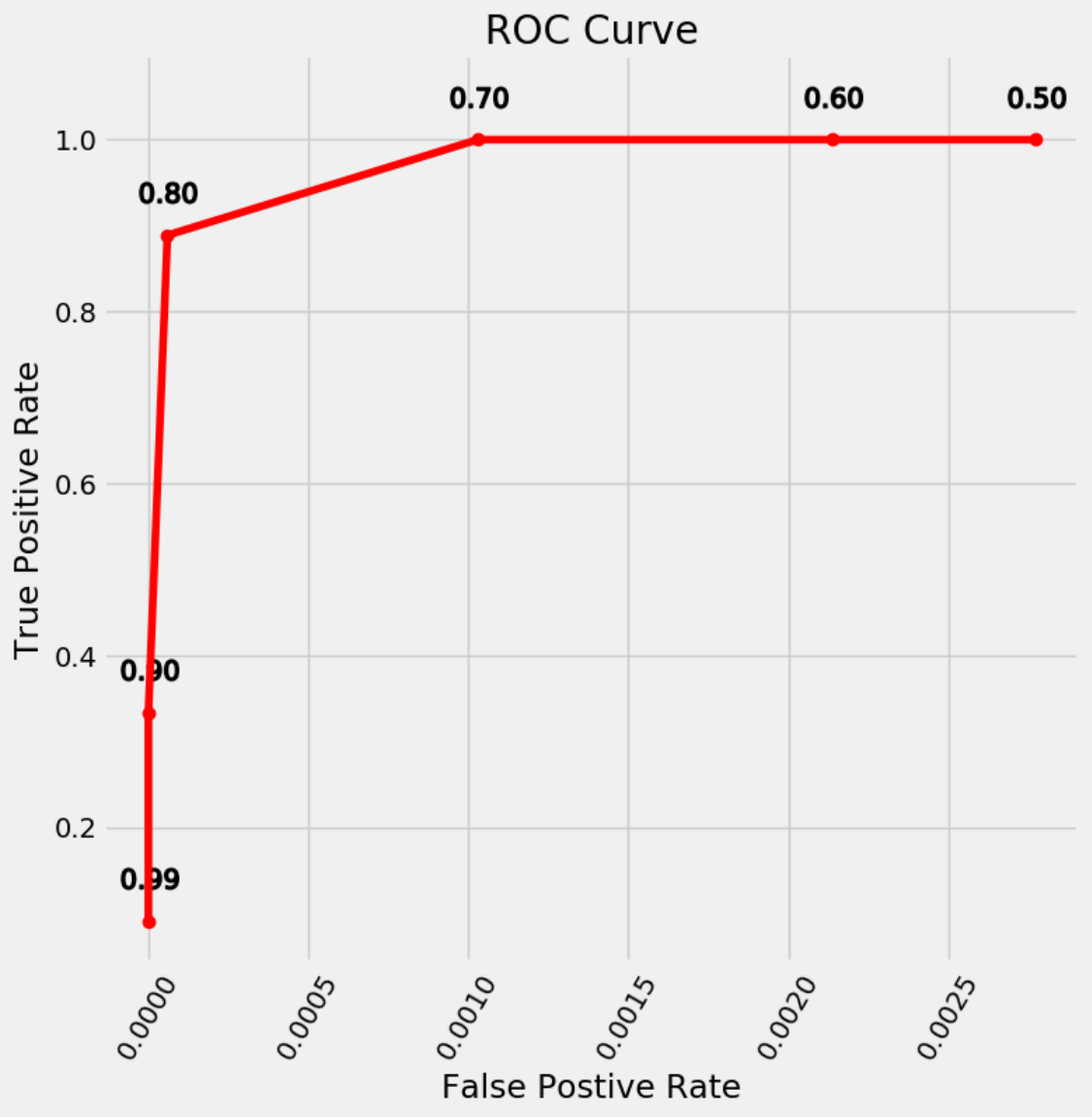
```
In [19]: threshold_list = [0.5, 0.6, 0.7, 0.8, 0.9, 0.99]

         tp_list = [9, 9, 9, 8, 3, 1]
         fp_list = [140, 108, 52, 3, 0, 0]
         tn_list = [50535 - 149, 50535 - 117, 50535 - 61,
                    50535 - 11, 50535 - 3, 50535 - 1]
         fn_list = [0, 0,  0, 1, 6, 10]
```

1.3 Function to Plot the ROC Curve

```
In [20]: def analyze_results(tp_list, fp_list, tn_list, fn_list, threshold_list,
                             actual_negatives, actual_positives):

             tpr_list = []
             fpr_list = []
             plt.style.use('fivethirtyeight')
             plt.figure(figsize=(8, 8))
             plt.xticks(rotation=60)

             for tp, fp, tn, fn, threshold in zip(tp_list, fp_list,
                                                  tn_list, fn_list,
                                                  threshold_list):

                 if fp == 0:
                     fpr = 0
                 else:
                     fpr = fp / (fp + tn)

                 tpr = tp / (tp + fn)

                 tpr_list.append(tpr)
                 fpr_list.append(fpr)

                 plt.plot(fpr, tpr + 0.05, marker='$%.2f$' % threshold,
                          ms = 30, color = 'k',
                         label = 'threshold')

             plt.plot(fpr_list, tpr_list, 'o-', color = 'red');
```

```
    plt.xlabel('False Postive Rate'); plt.ylabel('True Positive Rate');
    plt.title('ROC Curve');
    plt.show()
```

```
In [21]: analyze_results(tp_list, fp_list, tn_list, fn_list, threshold_list,
                         actual_negatives=50535, actual_positives=9)
```



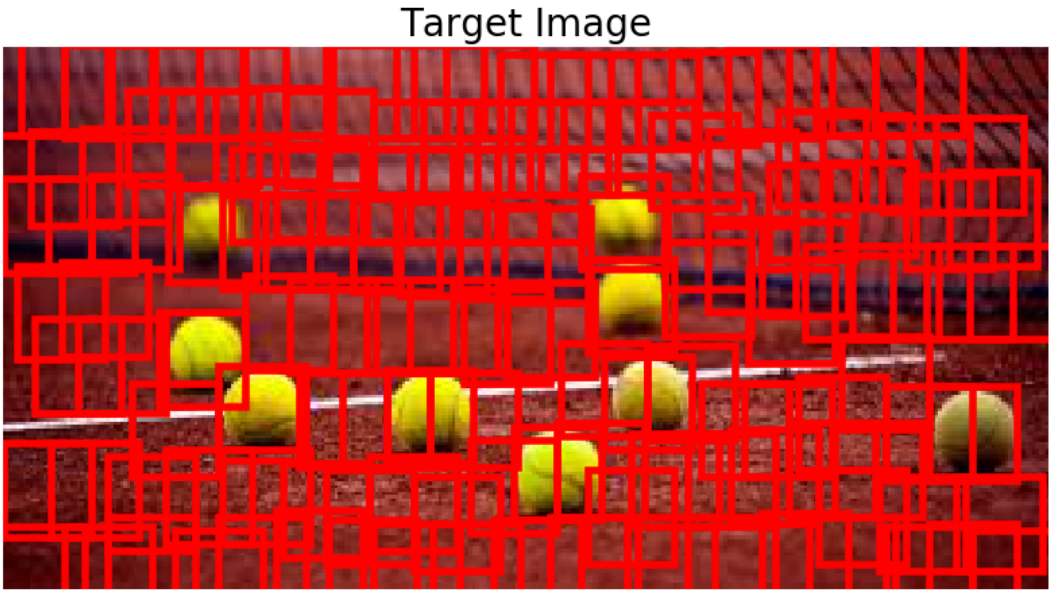
1.4 Shifting the Curve

Altering the threshold shifts the location along one ROC curve. This does not change the signal to noise ratio though, so this approach will only move along the curve. To shift the curve requires changing the signal to noise ratio.

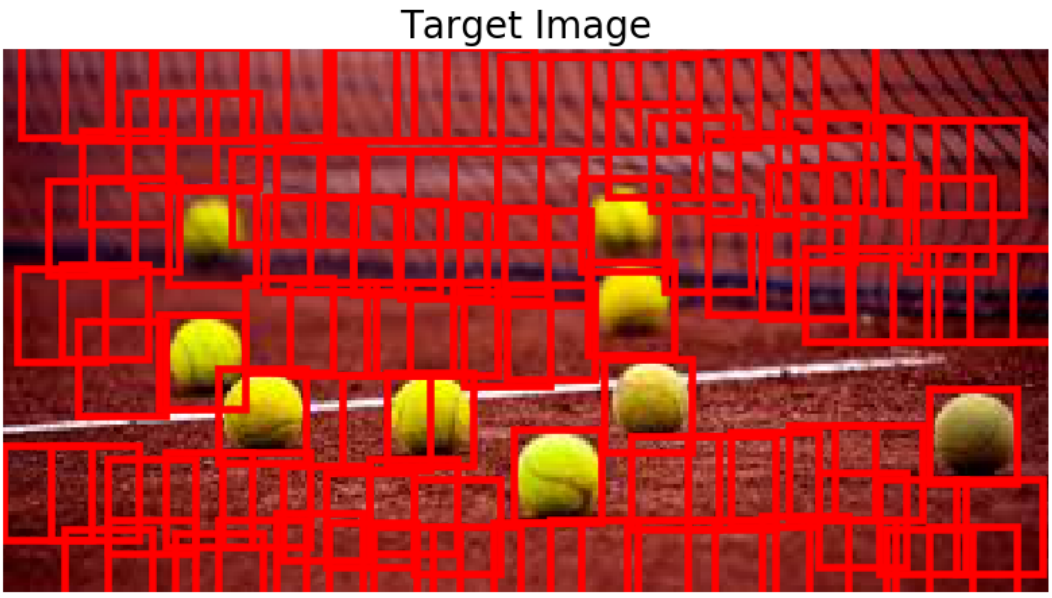
One way to shift the curve would be to adjust how far apart boxes have to be in the method, or using a different template. I will choose another of the images for the template. Both of these approaches change the signal to noise ratio. Changning the template image can increase or decrease the signal depending on the quality of the template.

```
In [7]: threshold_list = [0.5, 0.6, 0.7, 0.8, 0.9, 0.99]

        for threshold in threshold_list:
            model(image_path='images/target_tennis.jpg',
                  template_path='images/template_tennis_two.jpg',
                  threshold=threshold, analyze=True)
```

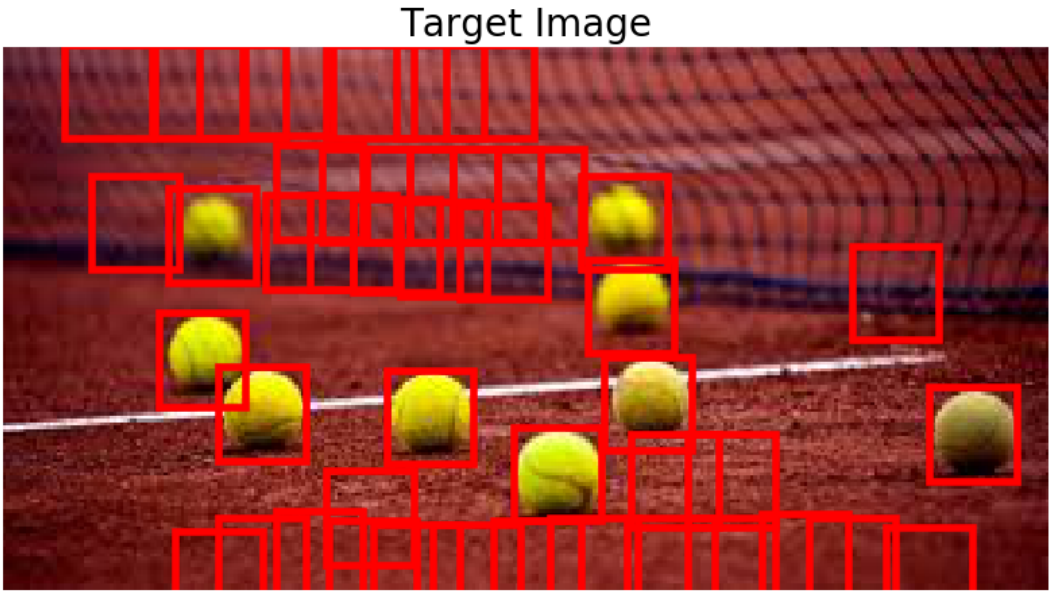
```
133 detections with threshold: 0.5
```



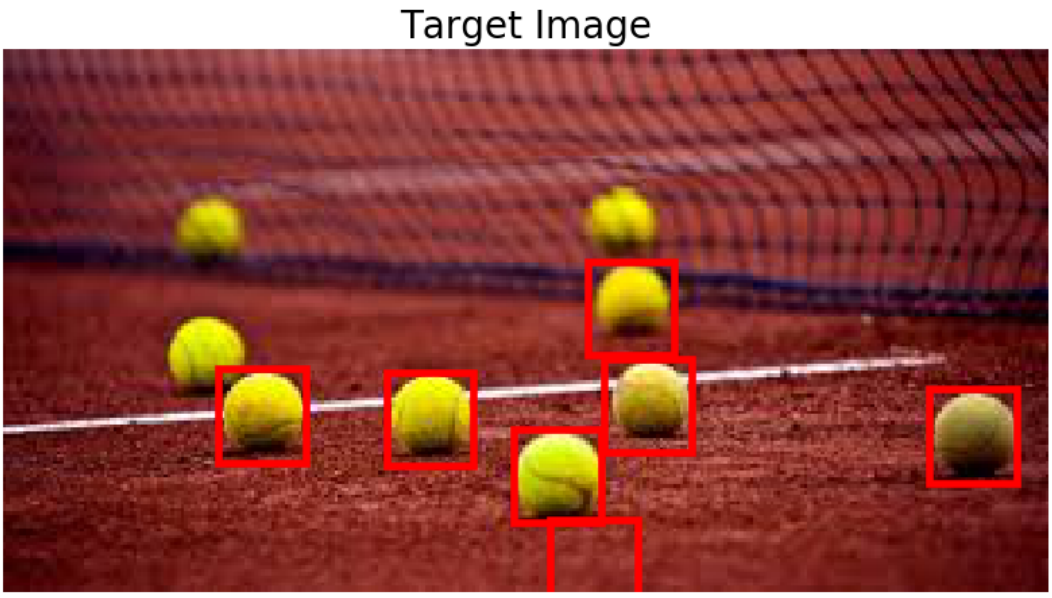

```
102 detections with threshold: 0.6
```



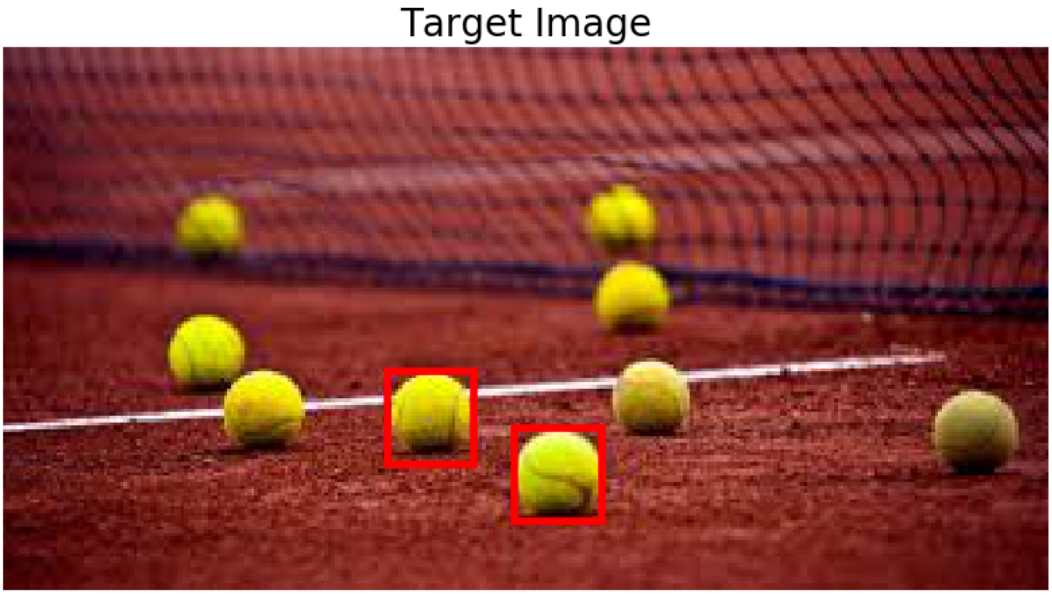

```
45 detections with threshold: 0.7
```



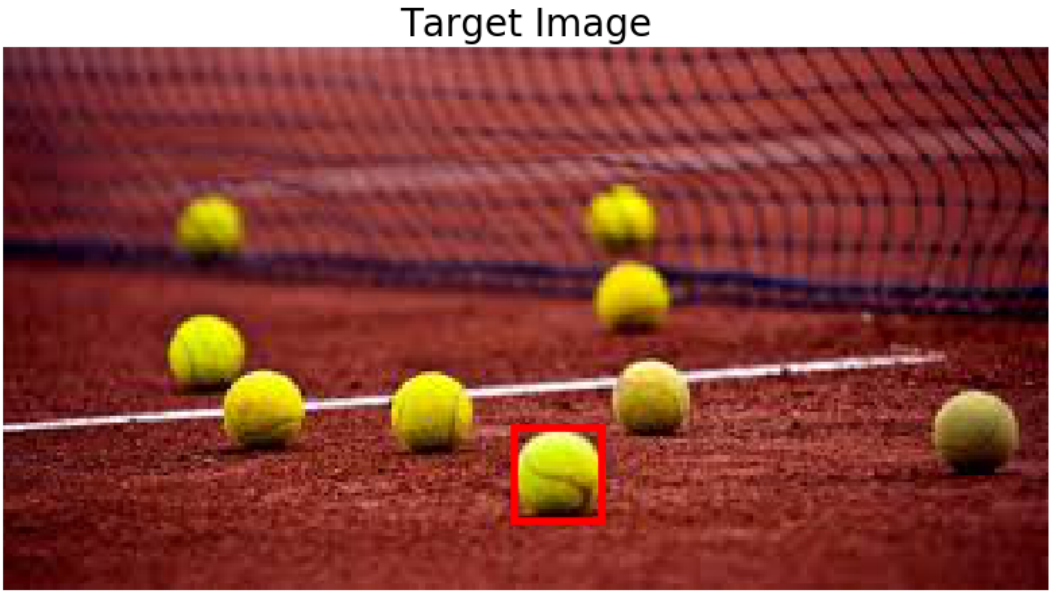

```
7 detections with threshold: 0.8
```



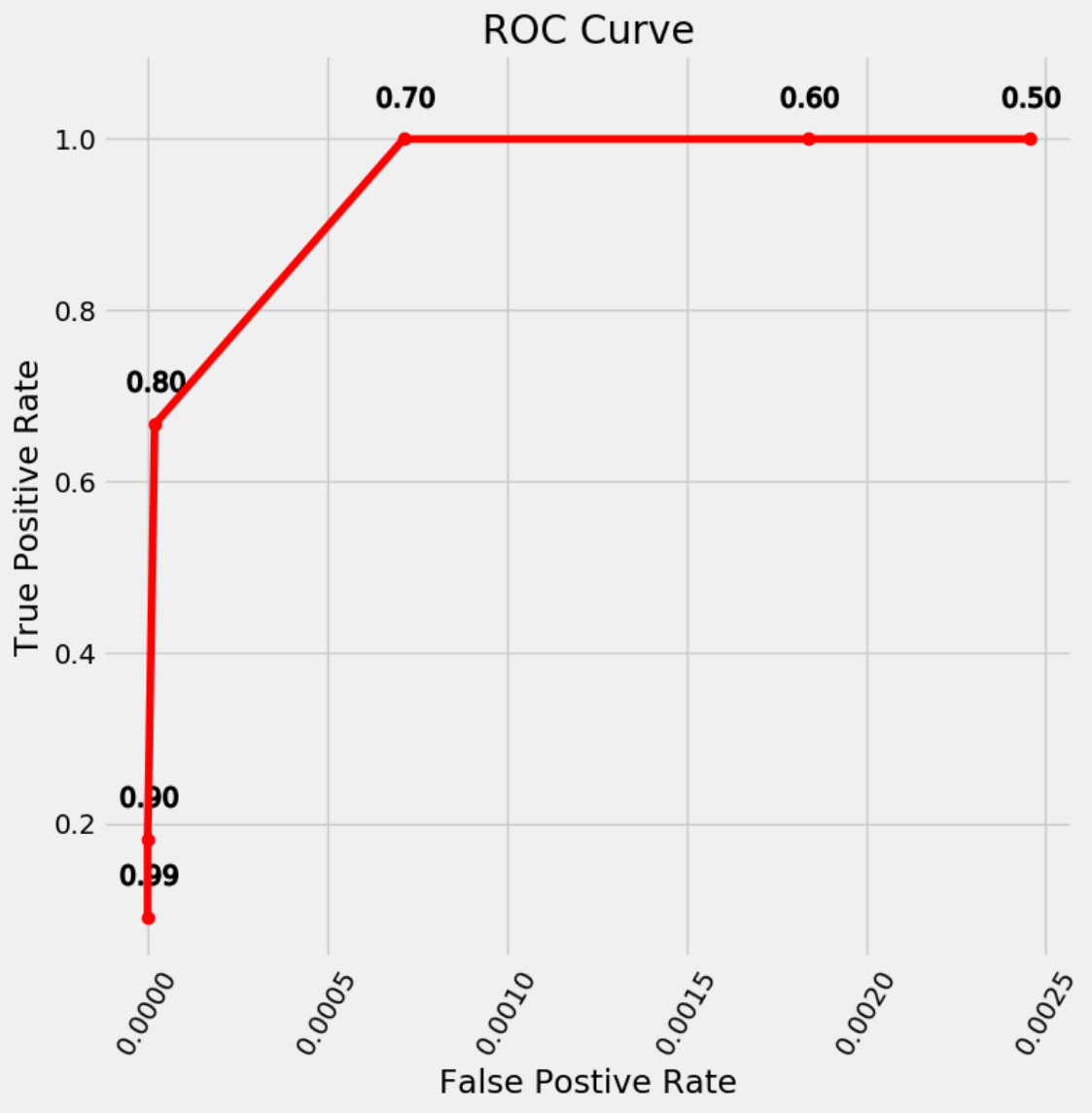

```
2 detections with threshold: 0.9
```




```
1 detections with threshold: 0.99
```




```
In [23]: tp_list = [9, 9, 9, 6, 2, 1]
         fp_list = [124, 93, 36, 1, 0, 0]
         tn_list = [50535 - 133, 50535 - 102, 50535 - 45,
                    50535 - 7, 50535 - 2, 50535 - 1]
         fn_list = [0, 0,  0, 3, 9, 10]

In [24]: analyze_results(tp_list, fp_list, tn_list, fn_list,
                         threshold_list, actual_negatives=50535, actual_positives=9)
```



1.5 Blur the Image before Detecting Features

Another approach to shift the curve is to blur the image using a Gaussian kernel before detecting images. The Gaussian blur decreases the amount of noise in the image, so we would expect the ROC curve to shift up and to the left with the pre-processing step.

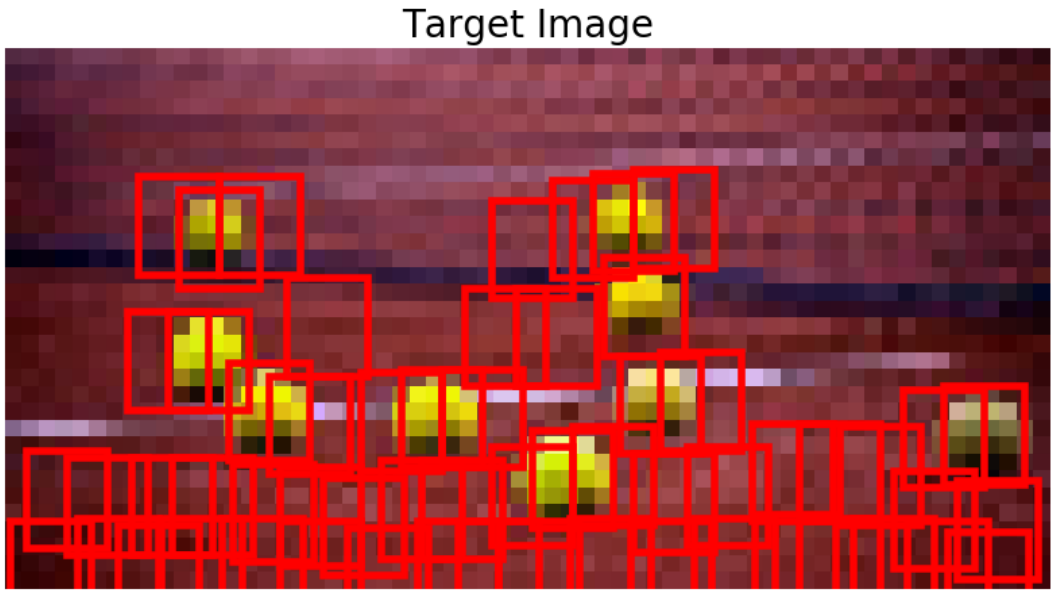
```
In [15]: from PIL import Image
         import matplotlib as mpl

         def apply_blur_filter(blur_filter, image_path):

             # Load in the image
```

```
image = Image.open(image_path)

# Crop to correct size
image = image.crop(box=(0, 0,
                        int(image.size[0] / blur_filter.shape[0]) * blur_filter.shape[0],
                        int(image.size[1] / blur_filter.shape[1]) * blur_filter.shape[1]))

im_array = np.array(image)

# Horizontal and vertical moves, using a stride of filter shape
h_moves = int(im_array.shape[1] / blur_filter.shape[1])
v_moves = int(im_array.shape[0] / blur_filter.shape[0])

new_image = np.zeros(shape = im_array.shape)

k = np.sum(blur_filter)

# Iterate through 3 color channels
for i in range(im_array.shape[2]):
    # Extract the layer and create a new layer to fill in
    layer = im_array[:, :, i]
    new_layer = np.zeros(shape = layer.shape, dtype='uint8')

    # Left and right bounds are determined by columns
    l_border = 0
    r_border = blur_filter.shape[1]


    # Iterate through the number of horizontal and vertical moves
    for h in range(h_moves):
        # Top and bottom bounds are determined by rows
        b_border = 0
        t_border = blur_filter.shape[0]
        for v in range(v_moves):
            patch = layer[b_border:t_border, l_border:r_border]

            # Take the element-wise product of the patch and the filter
            product = np.multiply(patch, blur_filter)

            # Find the weighted average of the patch
            product = np.sum(product) / k
            new_layer[b_border:t_border, l_border:r_border] = product

            b_border = t_border
            t_border = t_border + blur_filter.shape[0]

        l_border = r_border
        r_border = r_border + blur_filter.shape[1]
```

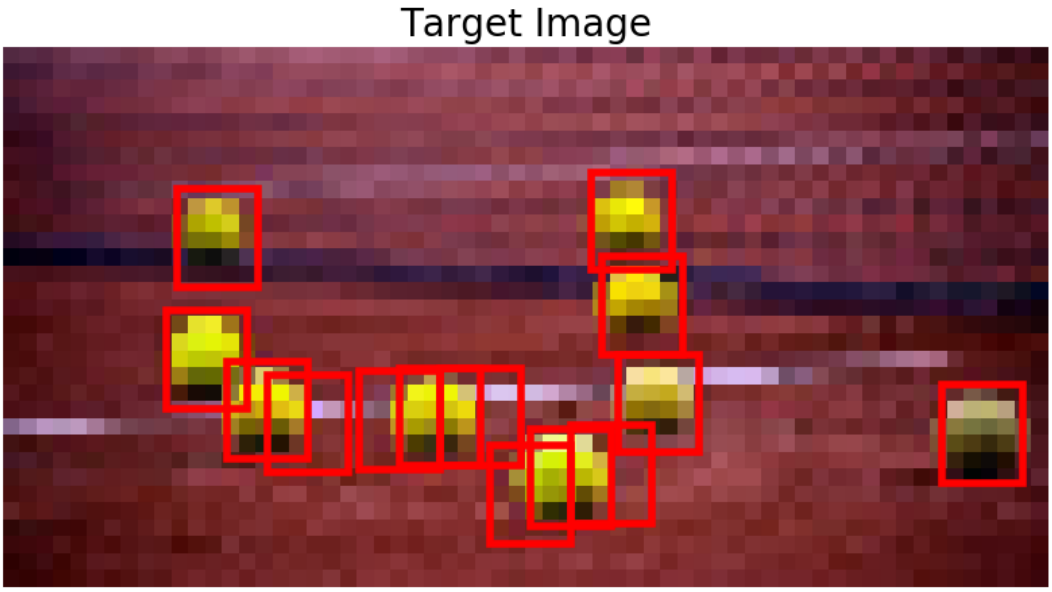
```
            new_image[:, :, i] = 255 * ( (new_layer - np.min(new_layer)) /
                                         (np.max(new_layer) - np.min(new_layer)) )


        # Convert to correct type for plotting
        new_image = new_image.astype('uint8')

        return new_image

    gaussian_kernel = np.array([[1, 4, 6, 4, 1],
                                [2, 8, 12, 8, 2],
                                [6, 24, 36, 24, 6],
                                [2, 8, 12, 8, 2],
                                [1, 4, 6, 4, 1]])

    blur_tennis = apply_blur_filter(gaussian_kernel, 'images/target_tennis.jpg')
    mpl.image.imsave('images/target_tennis_blurred.jpg', blur_tennis)
```

In [16]:
```
threshold_list = [0.5, 0.6, 0.7, 0.8, 0.9, 0.99]

for threshold in threshold_list:
    model(image_path='images/target_tennis_blurred.jpg',
          template_path='images/template_tennis.jpg',
          threshold=threshold, analyze=True)
```

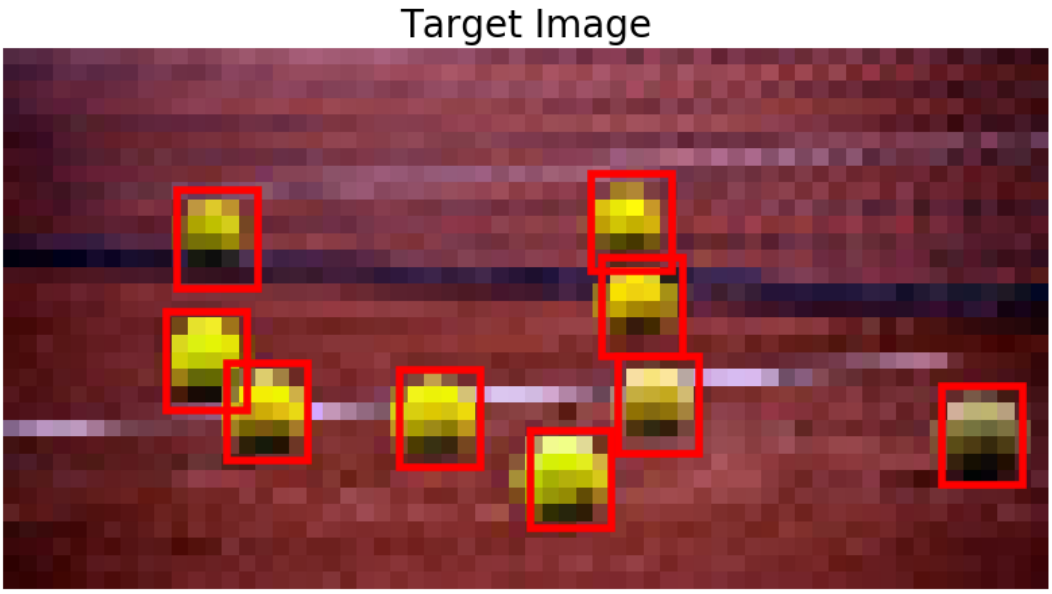
```
59 detections with threshold: 0.5
```



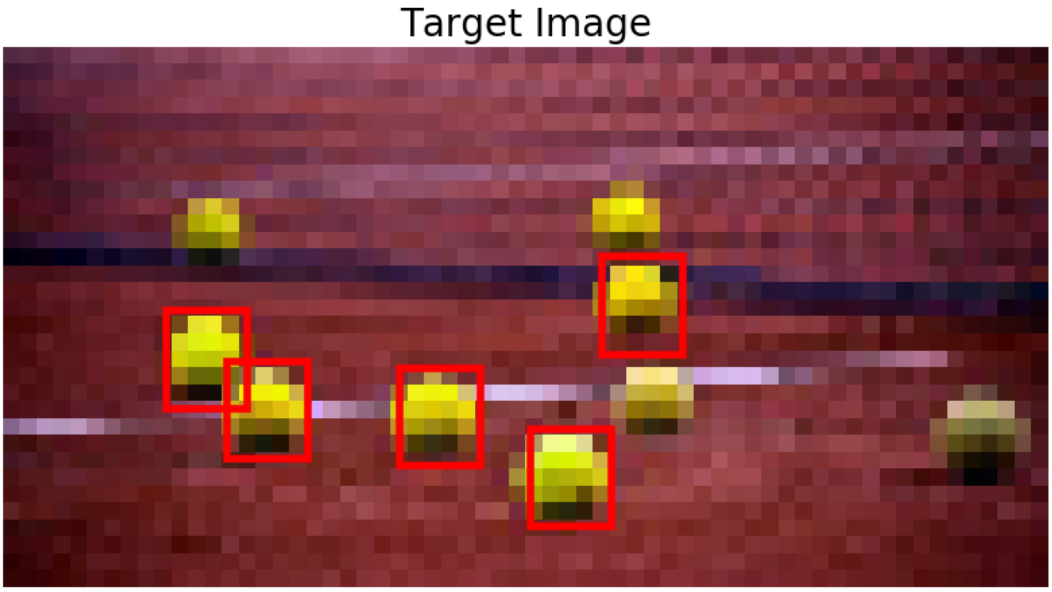
```
14 detections with threshold: 0.6
```



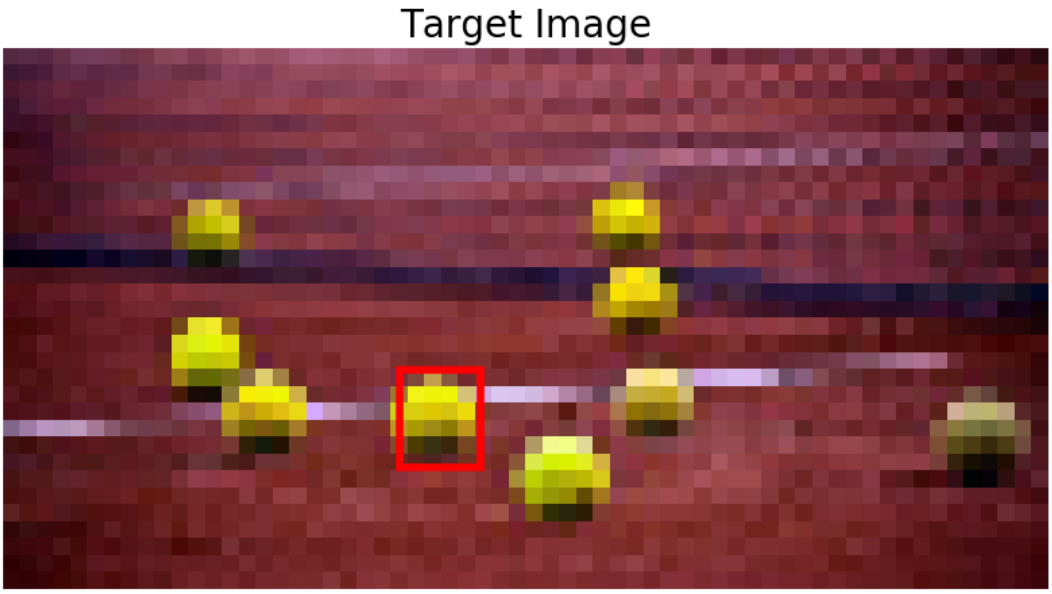

```
9 detections with threshold: 0.7
```



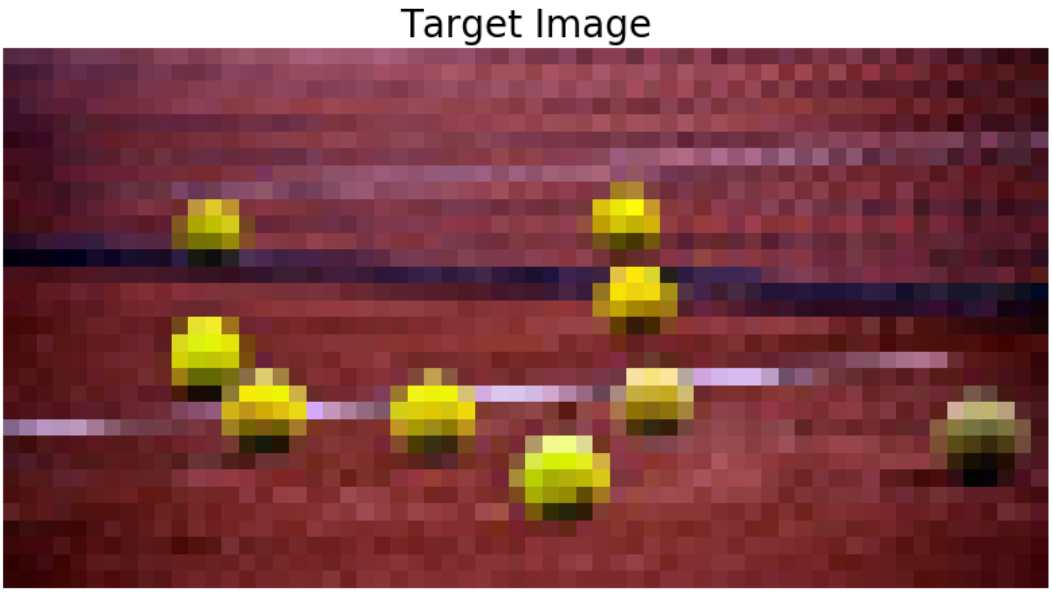
```
5 detections with threshold: 0.8
```



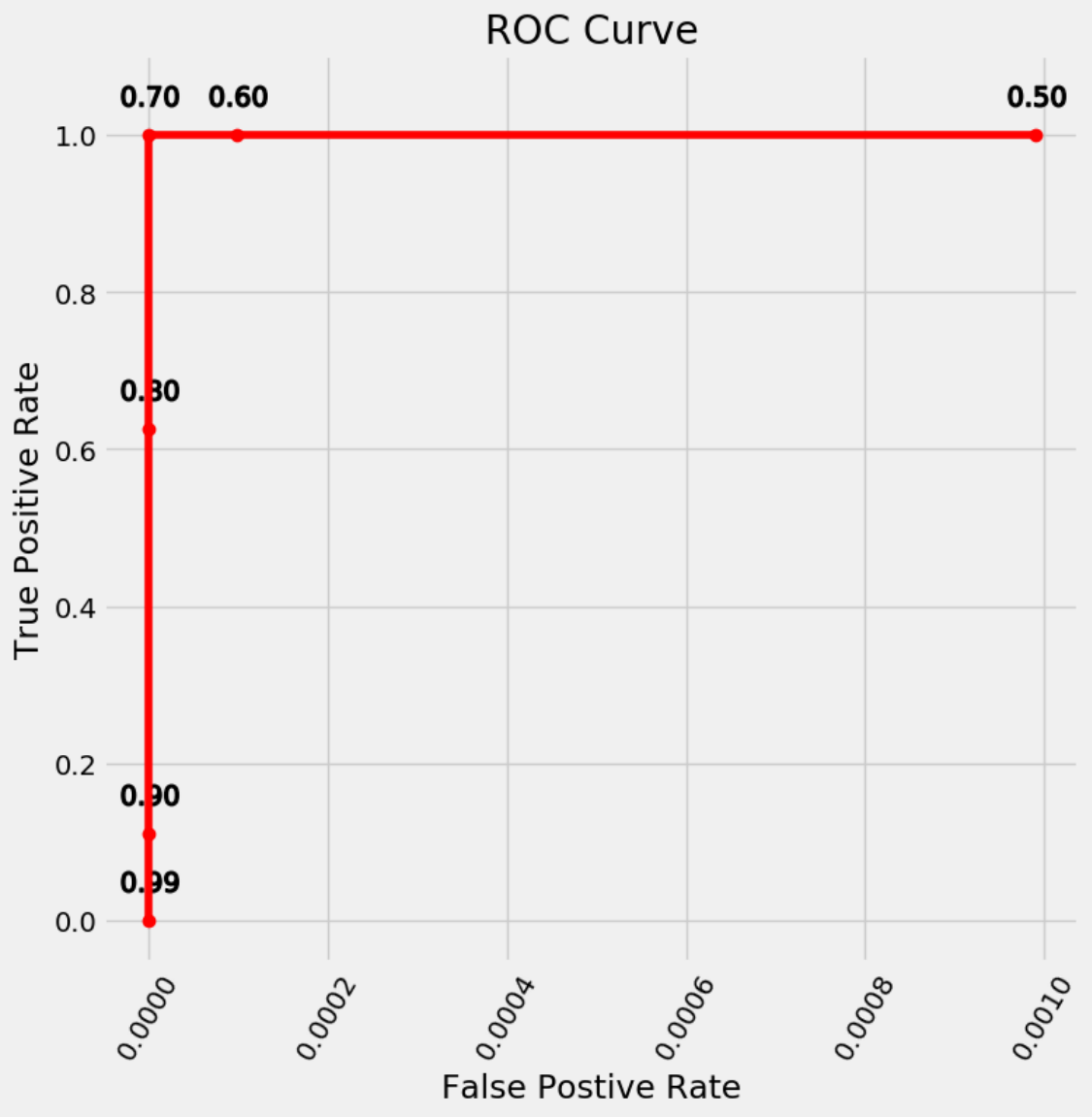

```
1 detections with threshold: 0.9
```



```
0 detections with threshold: 0.99
```




```
In [25]: threshold_list = [0.5, 0.6, 0.7, 0.8, 0.9, 0.99]

         tp_list = [9, 9, 9, 5, 1, 0]
         fp_list = [50, 5, 0, 0, 0, 0]
         tn_list = [50535 - 59, 50535 - 14, 50535 - 9,
                    50535 - 5, 50535 - 1, 50535]
         fn_list = [0, 0, 0, 3, 8, 9]

In [26]: analyze_results(tp_list, fp_list, tn_list, fn_list,
                         threshold_list, actual_negatives=50535, actual_positives=9)
```



Smoothing the image with a Gaussian Blur before feature detection and using a threshold of 0.7 results in a perfect classifier! The signal to noise ratio is changed by altering the content of the target image. This shows the benefits or pre-processing images using methods such as Gaussian Smoothing before perform image analysis.

2 Conclusions

In this notebook we created the Receiver Operating Characteristic Curve for the feature detection model. The ROC curve plots the true positive rate versus the false positive rate with a perfect classifier achieving a tpr of 1.0 and a fpr of 0.0. We can change the location along a given ROC curve be adjusting the threshold for establishing a detection based on the convolution results. However, to shift the entire curve requires changing the signal to noise ratio. We can do this by

using a different template, in this case a different image of the object we want to find. Another approach that works well (in this case making a perfect classifier) is applying pre-processing, such as a Gaussian Blur, to smooth the image. Smoothing the image with a Gaussian filter removes noise and details from the image resulting in fewer false positives. This notebook showed the idea of using the ROC to assess model performance, how to more along a given ROC curve, and using pre-processing to shift the entire curve.