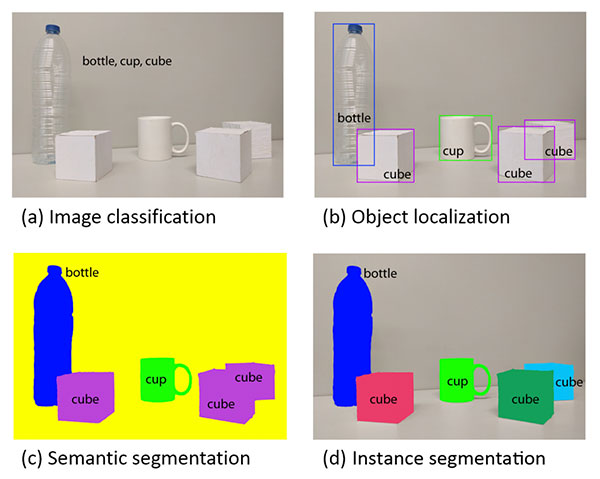
**Mask-RCNN**

**Instance segmentation**

**vs.**

**Semantic segmentation**

****

**Explaining the differences between traditional image classification, object detection, semantic segmentation, and instance segmentation is best done visually.**

**When performing traditional *image classification* our goal is to predict a set of labels to characterize the contents of an input image (*top-left*).**

***Object detection* builds on image classification, but this time allows us to localize each object in an image. The image is now characterized by:**

1. **Bounding box *(x, y)*-coordinates for each object**
2. **An associated *class label* for each bounding box**

**An example of *semantic segmentation* can be seen in *bottom-left*. Semantic segmentation algorithms require us to associate *every pixel* in an input image with a class label (including a class label for the background).**

**Pay close attention to our semantic segmentation visualization — notice how each object is indeed segmented but each “cube” object has the same color.**

**While semantic segmentation algorithms are capable of labeling every object in an image they *cannot* differentiate between two objects of the same class.**

**This behavior is especially problematic if two objects of the same class are partially occluding each other — we have no idea where the boundaries of one object ends and the next one begins, as demonstrated by the two purple cubes, we cannot tell where one cube starts and the other ends.**

***Instance segmentation* algorithms, on the other hand, compute a pixel-wise mask for every object in the image, even if the objects are of the same class label (*bottom-right*). Here you can see that each of the cubes has their own unique color, implying that our instance segmentation algorithm not only localized each individual cube but predicted their boundaries as well.**

**The Mask R-CNN architecture we’ll be discussing in this tutorial is an example of an *instance segmentation* algorithm.**

### What is Mask R-CNN?

**The Mask R-CNN algorithm was introduced by He et al. in their 2017 paper,**[**Mask R-CNN**](https://arxiv.org/abs/1703.06870)**.**

**Mask R-CNN builds on the previous object detection work of**[**R-CNN**](https://arxiv.org/abs/1311.2524)**(2013),**[**Fast R-CNN**](https://arxiv.org/abs/1504.08083)**(2015), and**[**Faster R-CNN**](https://arxiv.org/abs/1506.01497)**(2015), all by Girshick et al.**

**In order to understand Mask R-CNN let’s briefly review the R-CNN variants, starting with the original R-CNN:**

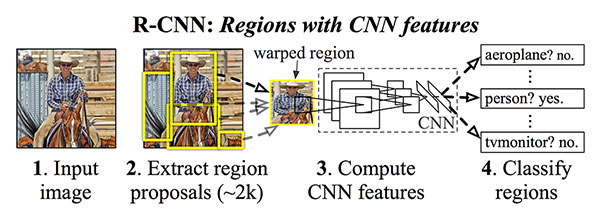
### 

### What is Mask R-CNN?

**The Mask R-CNN algorithm was introduced by He et al. in their 2017 paper,**[**Mask R-CNN**](https://arxiv.org/abs/1703.06870)**.**

**Mask R-CNN builds on the previous object detection work of**[**R-CNN**](https://arxiv.org/abs/1311.2524)**(2013),**[**Fast R-CNN**](https://arxiv.org/abs/1504.08083)**(2015), and**[**Faster R-CNN**](https://arxiv.org/abs/1506.01497)**(2015), all by Girshick et al.**

**In order to understand Mask R-CNN let’s briefly review the R-CNN variants, starting with the original R-CNN:**

****

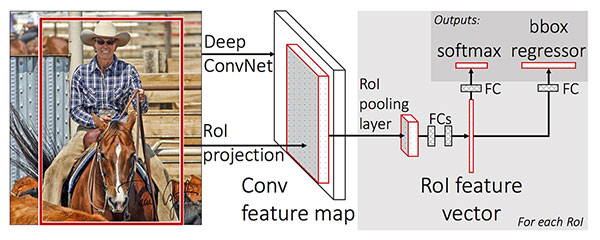
**The original R-CNN algorithm is a four-step process:**

* **Step #1: Input an image to the network.**
* **Step #2: Extract region proposals (i.e., regions of an image that potentially contain objects) using an algorithm such as**[**Selective Search**](http://www.huppelen.nl/publications/selectiveSearchDraft.pdf)**.**
* **Step #3: Use transfer learning, specifically feature extraction, to compute features for each proposal (which is effectively an ROI) using the pre-trained CNN.**
* **Step #4: Classify each proposal using the extracted features with a Support Vector Machine (SVM).**

**The reason this method works is due to the robust, discriminative features learned by the CNN.**

**However, the problem with the R-CNN method is it’s incredibly slow. And furthermore, we’re not actually learning to localize via a deep neural network, we’re effectively just building a more advanced**[**HOG + Linear SVM detector**](https://www.pyimagesearch.com/2014/11/10/histogram-oriented-gradients-object-detection/)**.**

**To improve upon the original R-CNN, Girshick et al. published the *Fast R-CNN* algorithm:**

****

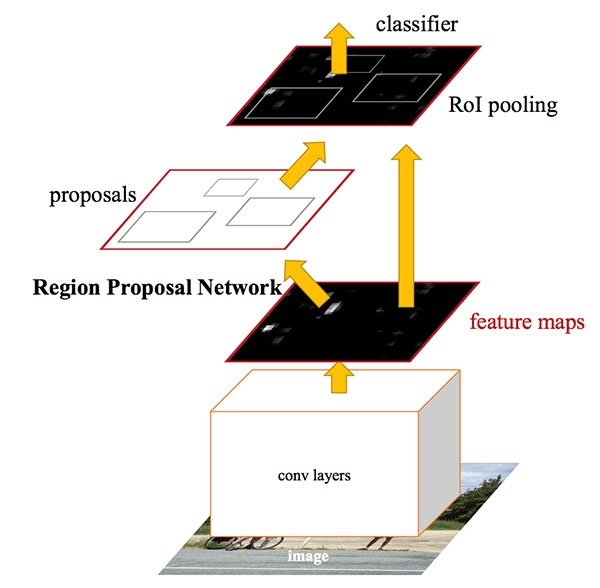
**Similar to the original R-CNN, Fast R-CNN still utilizes Selective Search to obtain region proposals; however, the novel contribution from the paper was Region of Interest (ROI) Pooling module.**

**ROI Pooling works by extracting a fixed-size window from the feature map and using these features to obtain the final class label and bounding box. The primary benefit here is that the network is now, effectively, end-to-end trainable:**

1. **We input an image and associated ground-truth bounding boxes**
2. **Extract the feature map**
3. **Apply ROI pooling and obtain the ROI feature vector**
4. **And finally, use the two sets of fully-connected layers to obtain (1) the class label predictions and (2) the bounding box locations for each proposal.**

**While the network is now end-to-end trainable, performance suffered dramatically at inference (i.e., prediction) by being dependent on Selective Search.**

**To make the R-CNN architecture even *faster* we need to incorporate the region proposal *directly* into the R-CNN:**

****

**The *Faster R-CNN* paper by Girshick et al. introduced the Region Proposal Network (RPN)that bakes region proposal *directly* into the architecture, alleviating the need for the Selective Search algorithm.**

**As a whole, the Faster R-CNN architecture is capable of running at approximately 7-10 FPS, a huge step towards making real-time object detection with deep learning a reality.**

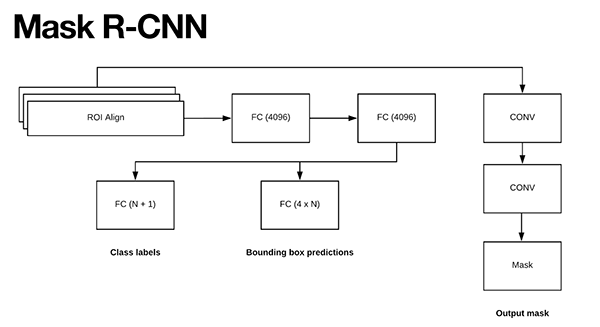
**The Mask R-CNN algorithm builds on the Faster R-CNN architecture with two major contributions:**

1. **Replacing the ROI Pooling module with a more accurate ROI Align module**
2. **Inserting an additional branch out of the ROI Align module**

**This additional branch accepts the output of the ROI Align and then feeds it into two CONV layers.**

**The output of the CONV layers is the mask itself.**

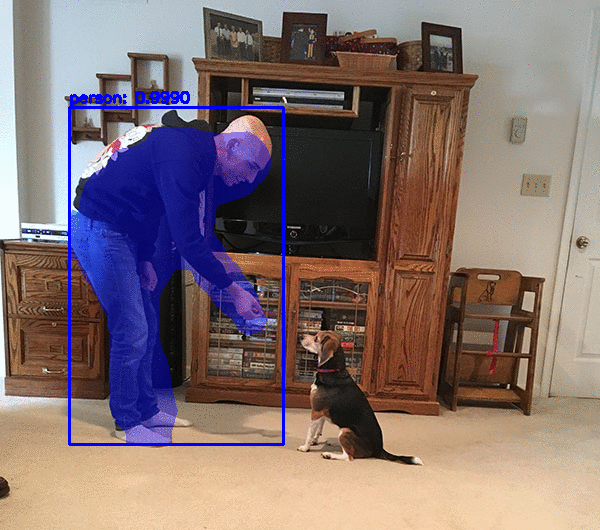
**We can visualize the Mask R-CNN architecture in the following figure:**

****

# **Mask R-CNN with OpenCV**

**by**[**Adrian Rosebrock**](https://www.pyimagesearch.com/author/adrian/)**on November 19, 2018 in**[**Deep Learning**](https://www.pyimagesearch.com/category/deep-learning-2/)**,**[**Semantic Segmentation**](https://www.pyimagesearch.com/category/semantic-segmentation/)**,**[**Tutorials**](https://www.pyimagesearch.com/category/tutorials/)

**[[](https://app.monstercampaigns.com/c/tortsem7qkvyuxc4cyfi/)Click here to download the source code to this post](https://app.monstercampaigns.com/c/tortsem7qkvyuxc4cyfi/)**

****

**In this tutorial, you will learn how to use Mask R-CNN with OpenCV.**

**Using Mask R-CNN you can automatically segment and construct pixel-wise masks for every object in an image. We’ll be applying Mask R-CNNs to both images and video streams.**

**In last week’s blog post you learned how to use the**[**YOLO object detector**](https://www.pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/)**to detect the presence of objects in images. Object detectors, such as YOLO, Faster R-CNNs, and Single Shot Detectors (SSDs), generate four sets of (x, y)-coordinates which represent the bounding box of an object in an image.**

**Obtaining the bounding boxes of an object is a good start but the bounding box itself doesn’t tell us anything about (1) which pixels belong to the foreground object and (2) which pixels belong to the background.**

**That begs the question:**

***Is it possible to generate a*mask*for each object in our image, thereby allowing us to*segment*the foreground object from the background?***

***Is such a method even possible?***

**The answer is yes — we just need to perform instance segmentation using the Mask R-CNN architecture.**

**To learn how to apply Mask R-CNN with OpenCV to both images and video streams, just keep reading!**

**Looking for the source code to this post?**[**Jump right to the downloads section.**](https://www.pyimagesearch.com/2018/11/19/mask-r-cnn-with-opencv/)

## **Mask R-CNN with OpenCV**

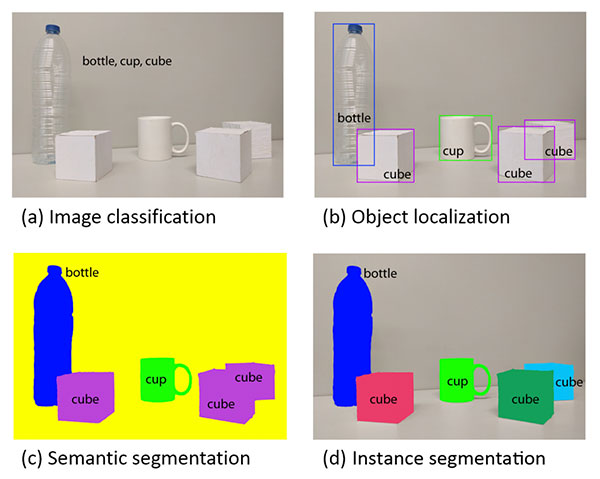
**In the first part of this tutorial, we’ll discuss the difference between image classification, object detection, instance segmentation, and semantic segmentation.**

**From there we’ll briefly review the Mask R-CNN architecture and its connections to Faster R-CNN.**

**I’ll then show you how to apply Mask R-CNN with OpenCV to both images and video streams.**

**Let’s get started!**

### Instance segmentation vs. Semantic segmentation

**[](https://www.pyimagesearch.com/wp-content/uploads/2018/11/mask_rcnn_segmentation_types.jpg)**

**Figure 1: Image classification (top-left), object detection (top-right), semantic segmentation (bottom-left), and instance segmentation (bottom-right). We’ll be performing instance segmentation with Mask R-CNN in this tutorial. (**[**source**](https://arxiv.org/abs/1704.06857)**)**

**Explaining the differences between traditional image classification, object detection, semantic segmentation, and instance segmentation is best done visually.**

**When performing traditional *image classification* our goal is to predict a set of labels to characterize the contents of an input image (top-left).**

***Object detection* builds on image classification, but this time allows us to localize each object in an image. The image is now characterized by:**

1. **Bounding box (x, y)-coordinates for each object**
2. **An associated class label for each bounding box**

**An example of *semantic segmentation* can be seen in bottom-left. Semantic segmentation algorithms require us to associate every pixel in an input image with a class label (including a class label for the background).**

**Pay close attention to our semantic segmentation visualization — notice how each object is indeed segmented but each “cube” object has the same color.**

**While semantic segmentation algorithms are capable of labeling every object in an image they cannot differentiate between two objects of the same class.**

**This behavior is especially problematic if two objects of the same class are partially occluding each other — we have no idea where the boundaries of one object ends and the next one begins, as demonstrated by the two purple cubes, we cannot tell where one cube starts and the other ends.**

***Instance segmentation* algorithms, on the other hand, compute a pixel-wise mask for every object in the image, even if the objects are of the same class label (bottom-right). Here you can see that each of the cubes has their own unique color, implying that our instance segmentation algorithm not only localized each individual cube but predicted their boundaries as well.**

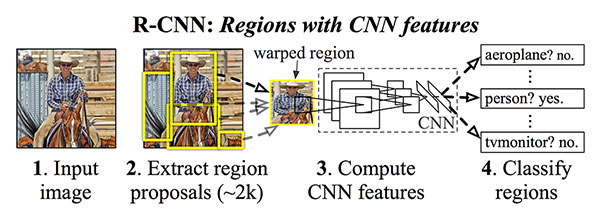
**The Mask R-CNN architecture we’ll be discussing in this tutorial is an example of an *instance segmentation* algorithm.**

### What is Mask R-CNN?

**The Mask R-CNN algorithm was introduced by He et al. in their 2017 paper,**[**Mask R-CNN**](https://arxiv.org/abs/1703.06870)**.**

**Mask R-CNN builds on the previous object detection work of**[**R-CNN**](https://arxiv.org/abs/1311.2524)**(2013),**[**Fast R-CNN**](https://arxiv.org/abs/1504.08083)**(2015), and**[**Faster R-CNN**](https://arxiv.org/abs/1506.01497)**(2015), all by Girshick et al.**

**In order to understand Mask R-CNN let’s briefly review the R-CNN variants, starting with the original R-CNN:**

**[](https://www.pyimagesearch.com/wp-content/uploads/2018/11/mask_rcnn_rcn_orig.jpg)**

**Figure 2: The original R-CNN architecture (source: [Girshick et al,. 2013](https://arxiv.org/abs/1311.2524" \t "blank))**

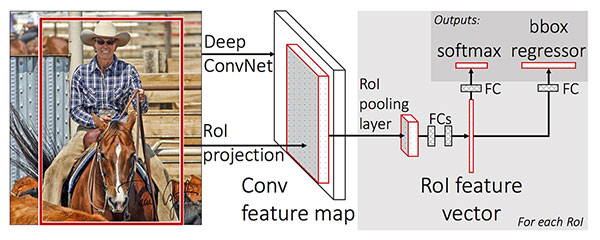
**The original R-CNN algorithm is a four-step process:**

* **Step #1: Input an image to the network.**
* **Step #2: Extract region proposals (i.e., regions of an image that potentially contain objects) using an algorithm such as**[**Selective Search**](http://www.huppelen.nl/publications/selectiveSearchDraft.pdf)**.**
* **Step #3: Use transfer learning, specifically feature extraction, to compute features for each proposal (which is effectively an ROI) using the pre-trained CNN.**
* **Step #4: Classify each proposal using the extracted features with a Support Vector Machine (SVM).**

**The reason this method works is due to the robust, discriminative features learned by the CNN.**

**However, the problem with the R-CNN method is it’s incredibly slow. And furthermore, we’re not actually learning to localize via a deep neural network, we’re effectively just building a more advanced**[**HOG + Linear SVM detector**](https://www.pyimagesearch.com/2014/11/10/histogram-oriented-gradients-object-detection/)**.**

**To improve upon the original R-CNN, Girshick et al. published the Fast R-CNN algorithm:**

**[](https://www.pyimagesearch.com/wp-content/uploads/2018/11/mask_rcnn_fast_rcnn.jpg)**

**Figure 3: The Fast R-CNN architecture (source: [Girshick et al., 2015](https://arxiv.org/abs/1504.08083" \t "blank)).**

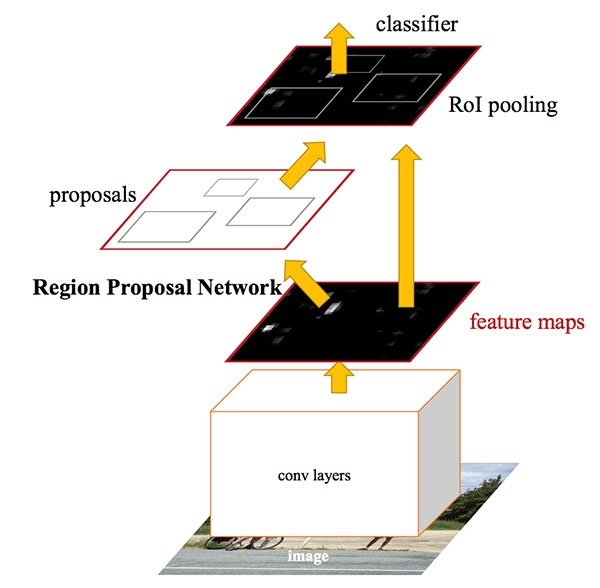
**Similar to the original R-CNN, Fast R-CNN still utilizes Selective Search to obtain region proposals; however, the novel contribution from the paper was Region of Interest (ROI) Pooling module.**

**ROI Pooling works by extracting a fixed-size window from the feature map and using these features to obtain the final class label and bounding box. The primary benefit here is that the network is now, effectively, end-to-end trainable:**

1. **We input an image and associated ground-truth bounding boxes**
2. **Extract the feature map**
3. **Apply ROI pooling and obtain the ROI feature vector**
4. **And finally, use the two sets of fully-connected layers to obtain (1) the class label predictions and (2) the bounding box locations for each proposal.**

**While the network is now end-to-end trainable, performance suffered dramatically at inference (i.e., prediction) by being dependent on Selective Search.**

**To make the R-CNN architecture even faster we need to incorporate the region proposal directly into the R-CNN:**

**[](https://www.pyimagesearch.com/wp-content/uploads/2018/11/mask_rcnns_faster_rcnn.jpg)**

**Figure 4: The Faster R-CNN architecture (source: [Girshick et al., 2015](https://arxiv.org/abs/1506.01497" \t "blank))**

**The Faster R-CNN paper by Girshick et al. introduced the Region Proposal Network (RPN)that bakes region proposal directly into the architecture, alleviating the need for the Selective Search algorithm.**

**As a whole, the Faster R-CNN architecture is capable of running at approximately 7-10 FPS, a huge step towards making real-time object detection with deep learning a reality.**

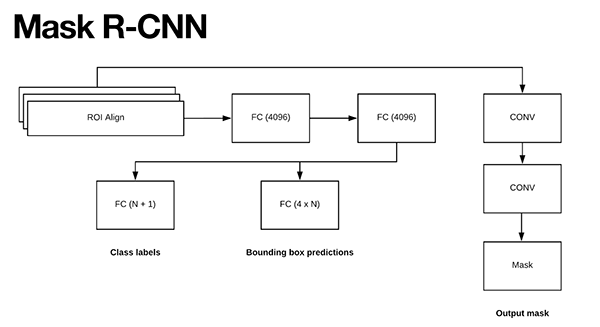
**The Mask R-CNN algorithm builds on the Faster R-CNN architecture with two major contributions:**

1. **Replacing the ROI Pooling module with a more accurate ROI Align module**
2. **Inserting an additional branch out of the ROI Align module**

**This additional branch accepts the output of the ROI Align and then feeds it into two CONV layers.**

**The output of the CONV layers is the mask itself.**

**We can visualize the Mask R-CNN architecture in the following figure:**

**[](https://www.pyimagesearch.com/wp-content/uploads/2018/11/mask_rcnn_arch.png)**

**Figure 5: The Mask R-CNN work by He et al. replaces the ROI Polling module with a more accurate ROI Align module. The output of the ROI module is then fed into two CONV layers. The output of the CONV layers is the mask itself.**

**Notice the branch of two CONV layers coming out of the ROI Align module — this is where our mask is actually generated.**

**As we know, the Faster R-CNN/Mask R-CNN architectures leverage a Region Proposal Network (RPN) to generate regions of an image that potentially contain an object.**

**Each of these regions is ranked based on their “objectness score” (i.e., how likely it is that a given region could potentially contain an object) and then the top N most confident objectness regions are kept.**

**In the original Faster R-CNN publication Girshick et al. set N=2,000, but in practice, we can get away with a much smaller N, such as N={10, 100, 200, 300} and still obtain good results.**

**He et al. set N=300 in**[**their publication**](https://arxiv.org/abs/1703.06870)**which is the value we’ll use here as well.**

**Each of the 300 selected ROIs go through three parallel branches of the network:**

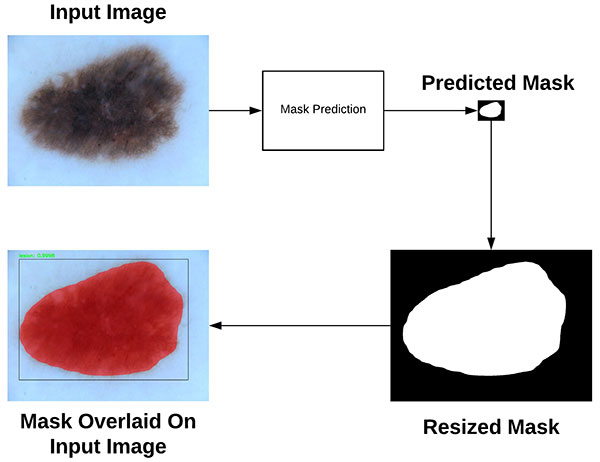
1. **Label prediction**
2. **Bounding box prediction**
3. **Mask prediction**

**Figure 5 above above visualizes these branches.**

**During prediction, each of the 300 ROIs go through**[**non-maxima suppression**](https://www.pyimagesearch.com/2014/11/17/non-maximum-suppression-object-detection-python/)**and the top 100 detection boxes are kept, resulting in a 4D tensor of 100 x L x 15 x 15 where L is the number of class labels in the dataset and 15 x 15 is the size of each of the L masks.**

**The Mask R-CNN we’re using here today was trained on the**[**COCO dataset**](http://cocodataset.org/#home)**, which has L=90classes, thus the resulting volume size from the mask module of the Mask R CNN is 100 x 90 x 15 x 15.**

**To visualize the Mask R-CNN process take a look at the figure below:**

****

**Here you can see that we start with our input image and feed it through our Mask R-CNN network to obtain our mask prediction.**

**The predicted mask is only *15 x 15* pixels so we resize the mask back to the original input image dimensions.**

**Finally, the resized mask can be overlaid on the original input image. For a more thorough discussion on how Mask R-CNN works be sure to refer to:**

1. **The original**[***Mask R-CNN***](https://arxiv.org/abs/1703.06870)**publication by He et al.**
2. **Adrian Rosebrock and his best selling book Deep Learning For Computer Vision.**