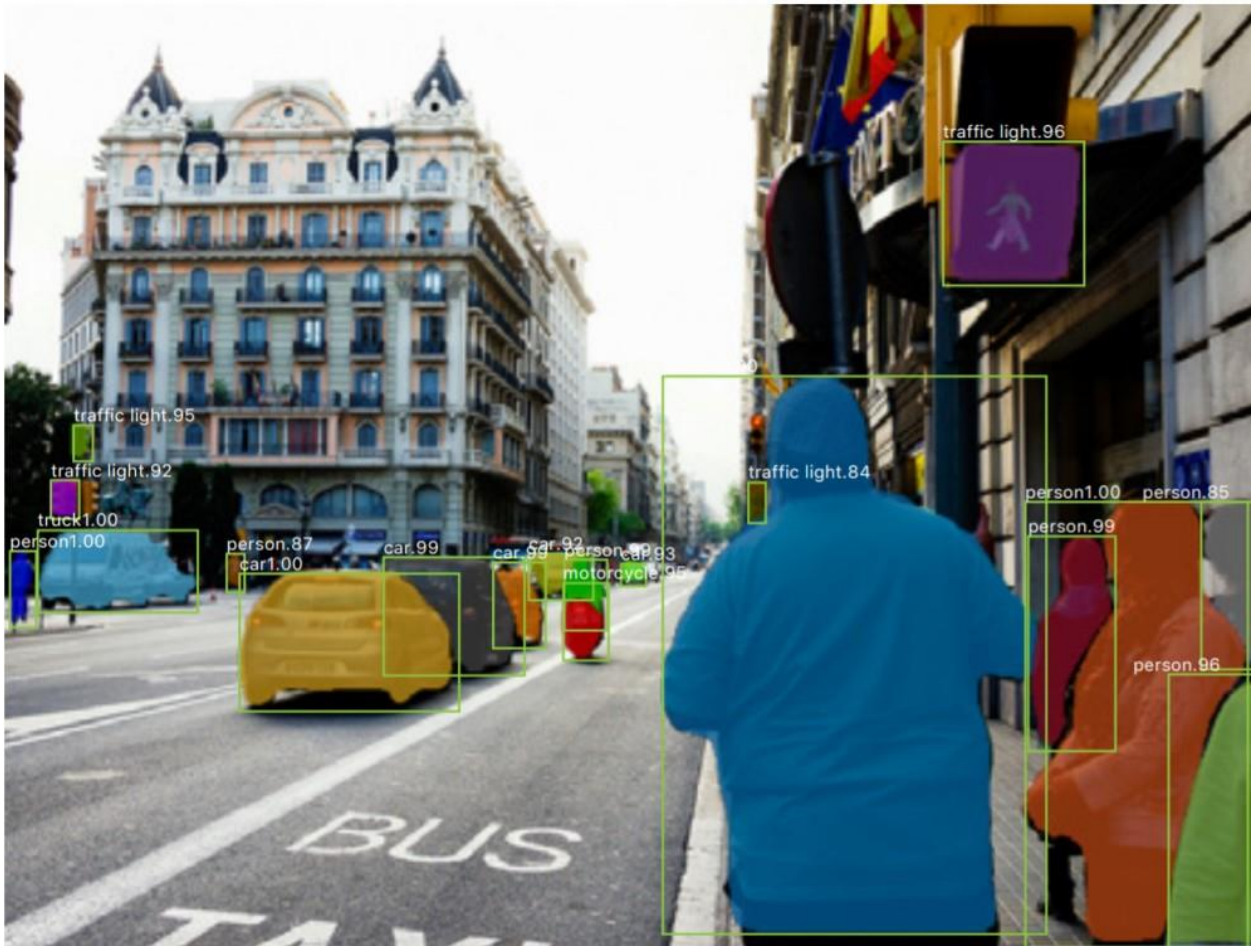
Image Segmentation with Machine Learning

Work on an intermediate-level Machine Learning Project – Image Segmentation
You might have wondered, how fast and efficiently our brain is trained to identify and classify what our eyes perceive. Somehow our brain is trained in a way to analyze everything at a granular level. This helps us distinguish an apple in a bunch of oranges.

Computer vision is a field of computer science that enables computers to identify and process objects in videos and images just the way we humans do. Although computer vision might seem like not a very old concept but it dates back to the late 1960s when the first digital image scanner which transformed images into grids of numbers was invented.



What is Image Segmentation?

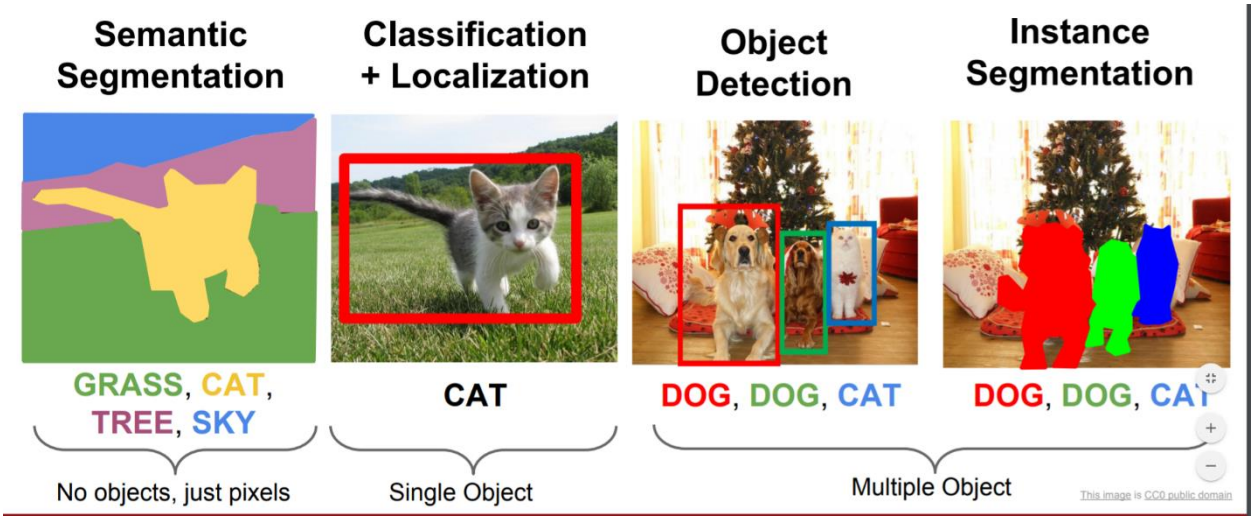
You would have probably heard about object detection and image localization. When there is a single object present in an image, we use image localization technique to draw a bounding box around that object. In the case of object detection, it provides labels along with the bounding boxes; hence we can predict the location as well as the class to which each object belongs.

Image segmentation results in more granular information about the shape of an image and thus an extension of the concept of Object Detection.

We segment i.e. divide the images into regions of different colors which helps in distinguishing an object from the other at a finer level

Types of Image Segmentation

Image Segmentation can be broadly classified into two types:



1. Semantic Segmentation

Semantic Segmentation is the process of segmenting the image pixels into their respective classes. For example, in the figure above, the cat is associated with yellow color; hence all the pixels related to the cat are colored yellow. Multiple objects of the same class are considered as a single entity and hence represented with the same color.

2. Instance Segmentation

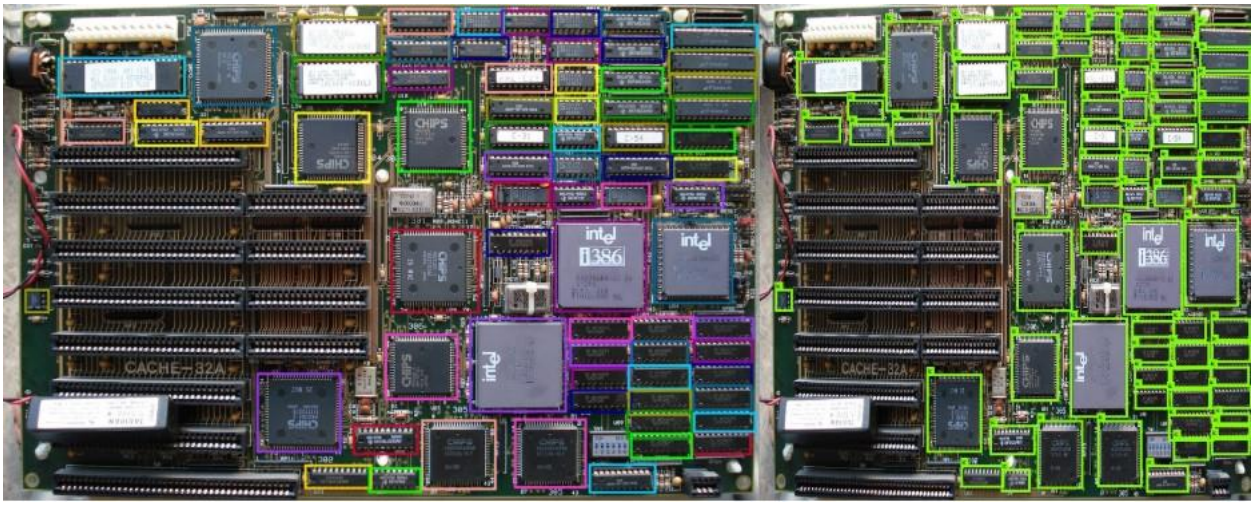
Instance segmentation is being more thorough and usually comes into picture when dealing with multiple objects. The difference here is, the detected object is masked with a color hence all the pixels associated with the image are given the same color. Multiple objects of the same class are treated as distinct entities and hence represented with different colors.

Image Segmentation Applications

1. Self-driving cars

Image segmentation can be used in self-driving cars for giving easy distinctions between various objects. Be it traffic signals, signboards, humans, and cars. It can help the driving instruction algorithm to better assess the surrounding before generating the next instruction.

2. Circuit Board Defect Detection



A company has to bear the responsibility of defected devices. If a camera backed with an Image Segmentation model keeps scanning for defects produced in the final product, a lot of money and time can be saved in fixing a defective device.

3. Face detection

Nowadays, we have observed that the majority of cameras in phones support portrait mode. Portrait mode is technically an outcome of Image Segmentation. Apart from this, security surveillance will be much more effective when the faces are distinguishable from noisy objects.

4. Medical Imaging

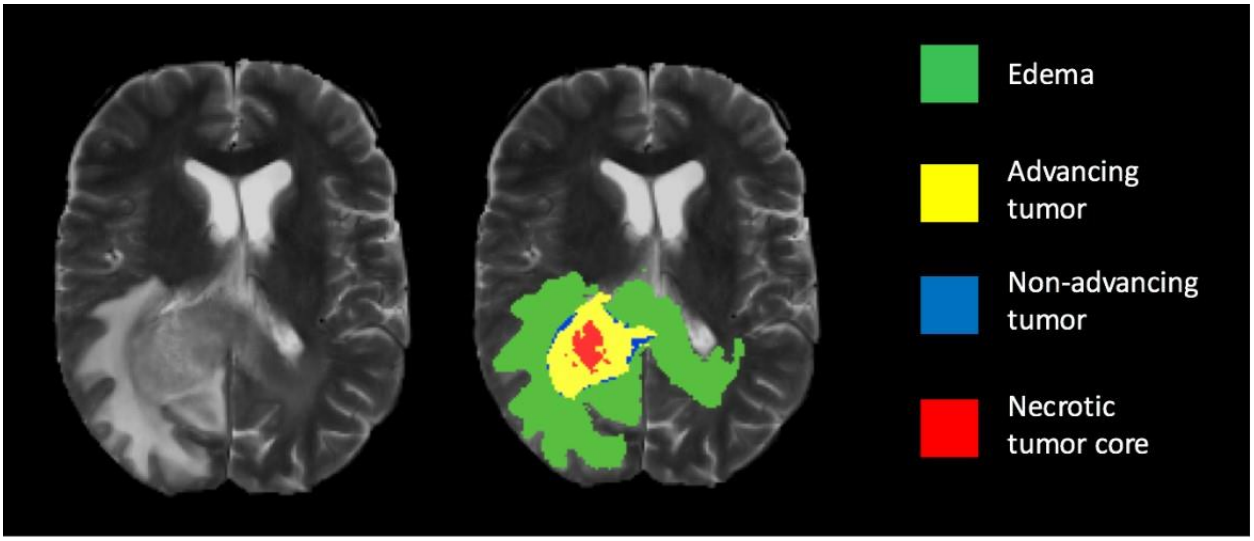


Image segmentation can be used to extract clinically relevant information from medical reports. For example, image segmentation can be used to segment tumors.

Mask R-CNN

We are going to perform image segmentation using the Mask R-CNN architecture. It is an extension of the Faster R-CNN Model which is preferred for object detection tasks.

The Mask R-CNN returns the binary object mask in addition to class label and object bounding box. Mask R-CNN is good at pixel level segmentation.

How does Mask R-CNN work?

Mask R-CNN uses an architecture similar to its predecessor Faster R-CNN and also utilizes Fully Convolutional Network for pixel-wise segmentation.

1. Feature Extraction

We utilize the ResNet 101 architecture to extract features from the input image. As a result, we get feature maps which are transmitted to Region Proposed Network

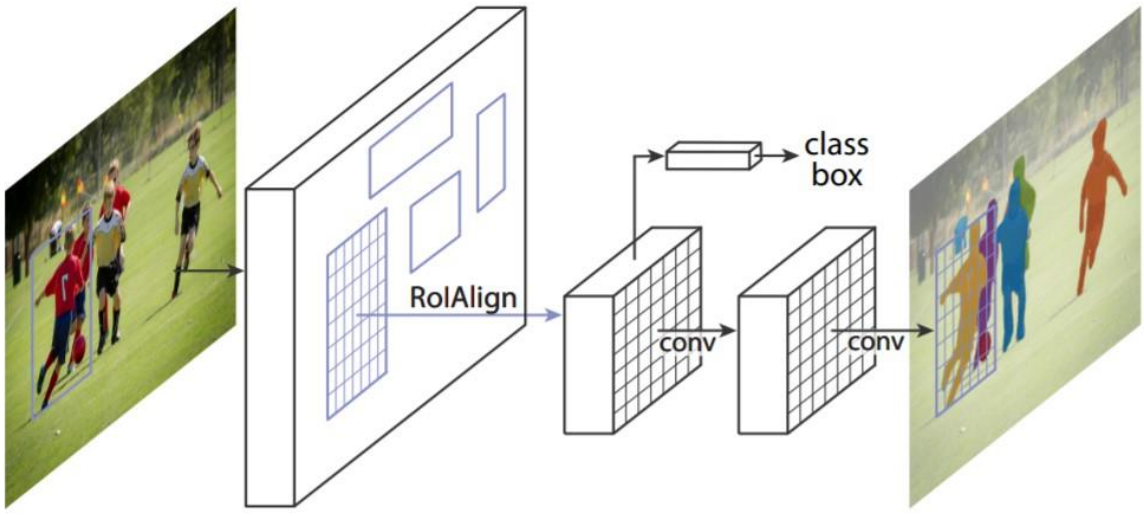
2. Region Proposed Network (RPN)

After obtaining the feature maps, bounding box candidates are determined and thus RPN extracts RoI (Region of Interest)

3. RoI Pool

Faster R-CNN uses an RoI Pool layer to compute features from the obtained proposals in order to infer the class of the object and bounding box coordinates.

4. RoI Align



RoI pool led to misalignments in getting the Region of Interest due to quantization of RoI coordinates. Since pixel-level segmentation required

specificity hence authors of the Faster R-CNN cleverly solved it by implementing the RoI Align.

Masking is done by a small fully-connected network applied to each RoI, which predicts a segmentation mask in a pixel-to-pixel manner.

Steps to develop Image Segmentation Project

Download Image Segmentation Project Code

Please download the source code of image segmentation: **Image Segmentation with Machine Learning**

1. Clone Mask R-CNN Github Repository

Now, primarily we download the architecture of the model which we are going to implement. Use the following command:

git clone: **https://github.com/matterport/Mask_RCNN.git**
Note : If you do not have git installed on your computer, then simply download the file in zip and extract the folder in your desired directory.

2. Library Dependencies

Now, since we need certain libraries in order to make it work as you might not have all the necessary libraries.
Here's the list,

- numpy
- scipy
- pillow
- cython
- matplotlib
- scikit-image
- tensorflow

- keras
- opencv-python
- h5py
- imgaug
- ipython

3. Pre Trained Weights

Since training a model takes hours and sometimes a day or more, hence it may not be feasible to train a model right now. Hence, we will utilize the pre-trained model to generate predictions on our input image.

Download Pretrained model from github

Follow this link, and you will see a list of the releases of Mask-RCNN. You can try for the latest release but since there were discrepancies, I have used **Mask R-CNN 2.0**. You can directly download the **h5 file** and save it in the samples folder of the Mask R-CNN repository we cloned in the first step.

4. Make a new Jupyter Notebook

So far, we have assembled the engine, it's time to utilize the power of our engine and drive all the way to our segmented image.

Now, we will make a new Jupyter Notebook under the samples folder in Mask R-CNN repository, you can use any other IDE but Jupyter Notebook gives the ease to execute code cell by cell.

If you do not have a powerful system, you can use **google colab** for running the code, but make sure to upload the repo and h5 file correctly.

5. Importing the Necessary Libraries

```
import os
import sys
import random
import math
import numpy as np
import skimage.io
import matplotlib
```

```
import matplotlib.pyplot as plt
# Fetching the root directory
ROOT_DIR = os.path.abspath("../")
import warnings
warnings.filterwarnings("ignore")
# Importing Mask RCNN
sys.path.append(ROOT_DIR) # To find local version of the library
from mrcnn import utils
import mrcnn.model as modellib
from mrcnn import visualize
# Heading to the coco directory
sys.path.append(os.path.join(ROOT_DIR, "samples/coco/"))
#importing coco.py
import coco
%matplotlib inline
```

Note 1: If you are confused or stuck in locating the directory, type in print (ROOT_DIR) to get the idea of the directory you are referring to.
Note 2: You might get an error associated with 'pycocotools'. In case you are unable to successfully install 'pycocotools' in 'Windows' then try to install this as it worked for me.

6. The path for pretrained weights

```
# Directory to save logs and trained model
MODEL_DIR = os.path.join(ROOT_DIR, "logs")
# Local path to trained weights file
COCO_MODEL_PATH = os.path.join(", "mask_rcnn_coco.h5")
# Directory of images to run detection on
DIR_IMAGE = os.path.join(ROOT_DIR, "images")
```

7. Inference class to infer the Mask R-CNN Model

```
class InferenceConfig(coco.CocoConfig):
# Setting batch size equal to 1 since we'll be running inference on
# one image at a time. Batch size = GPU_COUNT * IMAGES_PER_GPU
GPU_COUNT = 1
IMAGES_PER_GPU = 1
config = InferenceConfig()
config.display()
```

Output:

```
Configurations:
BACKBONE                resnet101
BACKBONE_STRIDES        [4, 8, 16, 32, 64]
BATCH_SIZE              1
BBOX_STD_DEV            [0.1 0.1 0.2 0.2]
COMPUTE_BACKBONE_SHAPE  None
DETECTION_MAX_INSTANCES 100
DETECTION_MIN_CONFIDENCE 0.7
DETECTION_NMS_THRESHOLD 0.3
FPN_CLASSIF_FC_LAYERS_SIZE 1024
GPU_COUNT               1
GRADIENT_CLIP_NORM      5.0
IMAGES_PER_GPU          1
IMAGE_CHANNEL_COUNT     3
IMAGE_MAX_DIM           1024
IMAGE_META_SIZE         93
IMAGE_MIN_DIM           800
IMAGE_MIN_SCALE         0
IMAGE_RESIZE_MODE       square
IMAGE_SHAPE             [1024 1024    3]
LEARNING_MOMENTUM       0.9
LEARNING_RATE           0.001
LOSS_WEIGHTS            {'rpn_class_loss': 1.0, 'rpn_bbox_loss': 1.0, 'mrcnn_class_loss': 1.0, 'mrcnn_bbox_loss': 1.0,
'mrcnn_mask_loss': 1.0}
MASK_POOL_SIZE          14
MASK_SHAPE              [28, 28]
MAX_GT_INSTANCES        100
MEAN_PIXEL              [123.7 116.8 103.9]
MINI_MASK_SHAPE         (56, 56)
NAME                    coco
NUM_CLASSES             81
POOL_SIZE               7
POST_NMS_ROIS_INFERENCE 1000
POST_NMS_ROIS_TRAINING  2000
PRE_NMS_LIMIT           6000
ROI_POSITIVE_RATIO      0.33
RPN_ANCHOR_RATIOS       [0.5, 1, 2]
RPN_ANCHOR_SCALES       (32, 64, 128, 256, 512)
RPN_ANCHOR_STRIDE       1
RPN_BBOX_STD_DEV        [0.1 0.1 0.2 0.2]
RPN_NMS_THRESHOLD       0.7
RPN_TRAIN_ANCHORS_PER_IMAGE 256
STEPS_PER_EPOCH         1000
TOP_DOWN_PYRAMID_SIZE   256
TRAIN_BN                False
TRAIN_ROIS_PER_IMAGE    200
USE_MINI_MASK           True
USE_RPN_ROIS            True
VALIDATION_STEPS        50
WEIGHT_DECAY            0.0001
```

What you are seeing is the specification of the Mask R-CNN model we are going to use. The backbone is **resnet101** which helps in extracting features from the image.
Next important thing to observe here is the mask shape which is 28×28 as it is trained on the COCO dataset and we have a total of 81 classes.

This means that there are 81 possible prediction classes in which an object may fall into.

8. Loading the Weights

```
# Create model objects in inference mode.
model = modellib.MaskRCNN(mode="inference", model_dir='mask_rcnn_coco.hy', config=config)
```

```
# Load weights trained on MS-COCO
model.load_weights('mask_rcnn_coco.h5', by_name=True)
```

9. Loading an Image to Test the Model

```
image = skimage.io.imread('../images/4410436637_7b0ca36ee7_z.jpg')
# original image
plt.figure(figsize=(12,10))
skimage.io.imshow(image)
```

Output:

```
<matplotlib.image.AxesImage at 0x1fc4b852b70>
```

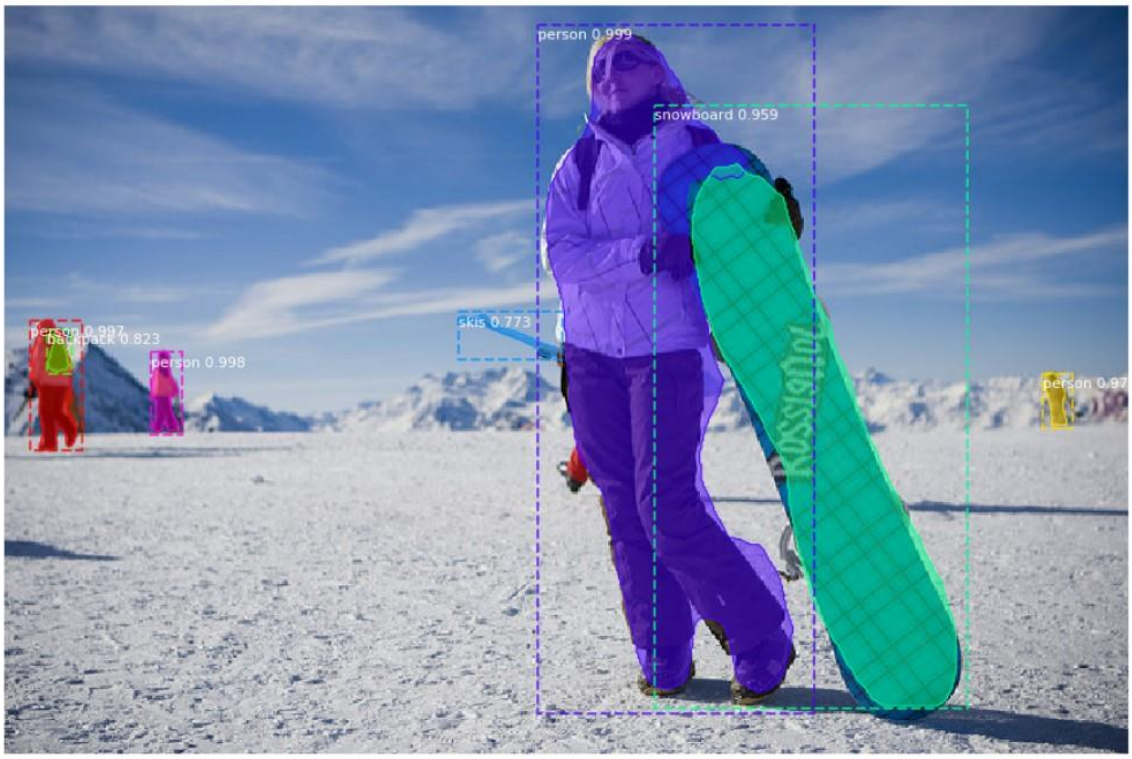
10. Sending Image to Model to Generate Predict

```
# Run detection
results = model.detect([image], verbose=1)
```

12. Masking the Results to our Image

```
# Visualize results
r = results[0]
visualize.display_instances(image, r['rois'], r['masks'], r['class_ids'],
```

The Time you were Desperately waiting, here comes our **Output:**



Hooray.! We have successfully segmented the image and we can see our code has performed pretty well. So cheers to you if you made it through.

13. Number of Detected Objects

Now, if you are curious, to detect the number of objects we were successfully able to detect, just type in

```
mask = r['masks']
mask = mask.astype(int)
mask.shape
```

Output:

(426, 640, 7)

Here, we can see that there are a total of 7 objects detected by our model on the image.

NOTE: Image & Video Source: Cornell University, Stanford University, Github

Summary

We hope we were able to lead you towards the solution of your first image segmentation problem. we discussed the process and if you are curious to know the details there is plenty of information available on the internet.

We would suggest you read and understand various architectures including the Mask R-CNN we implemented. It will help you analyze things better and also help you to generate new ideas to solve complex problems.