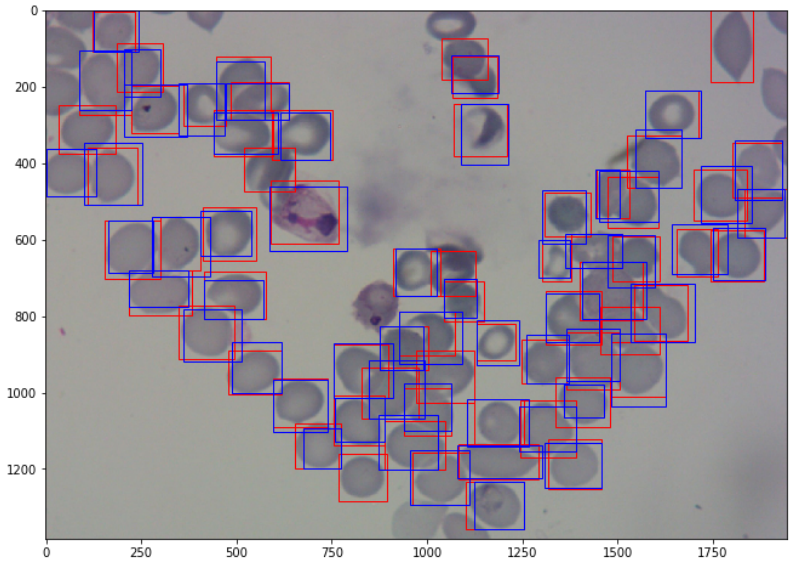
Object Detection on Malaria Images

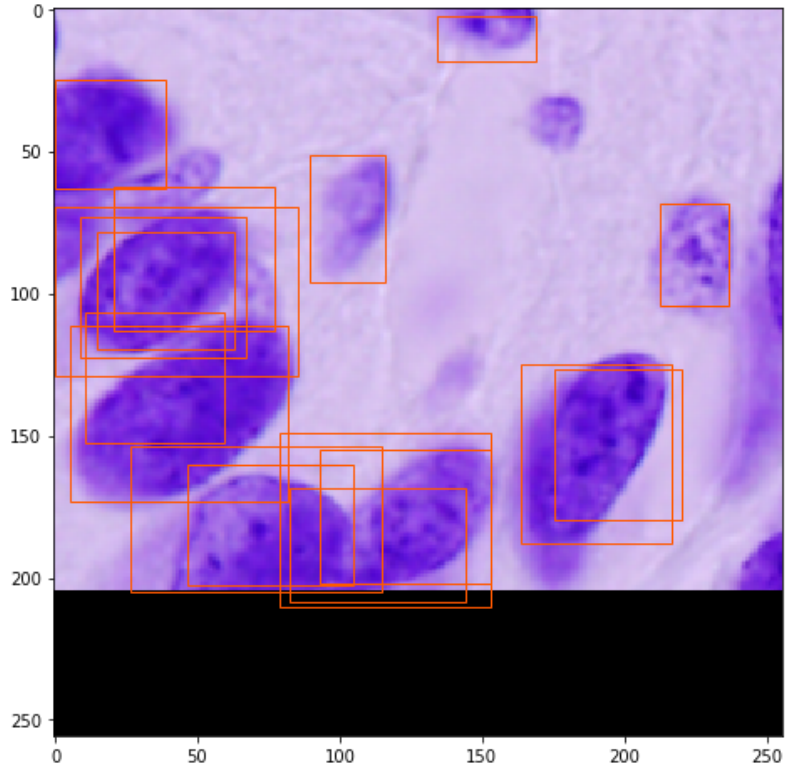
Description of the biological application : ¶

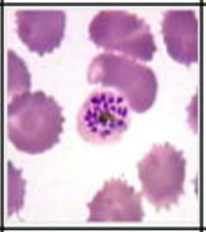
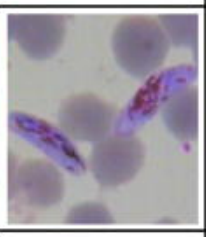
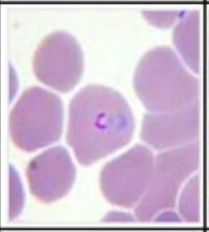
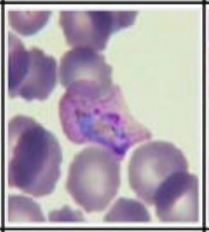
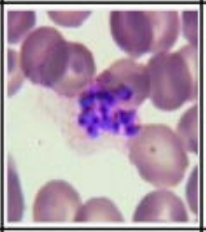
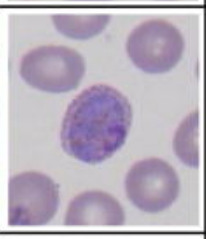
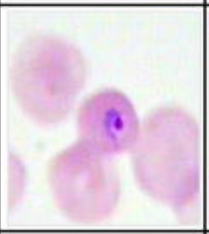
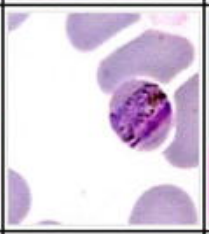
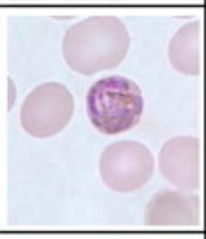
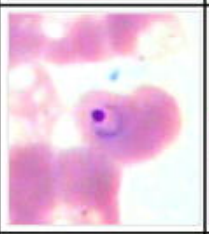
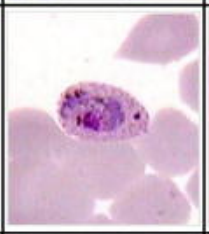
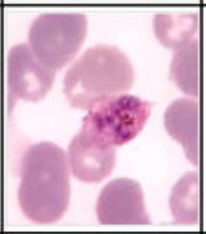
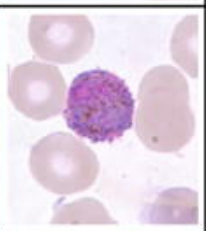
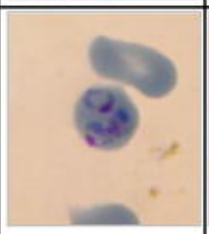
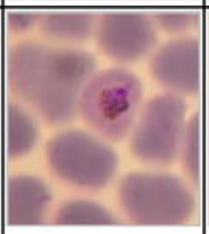
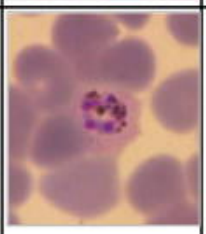
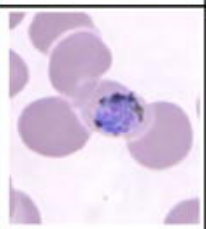
Malaria is a disease caused by Plasmodium parasites that remains a major threat in global health, affecting 200 million people and causing 400,000 deaths a year. The main species of malaria that affect humans are Plasmodium falciparum and Plasmodium vivax.
For malaria as well as other microbial infections, manual inspection of thick and thin blood smears by trained microscopists remains the gold standard for parasite detection and stage determination because of its low reagent and instrument cost and high flexibility. Despite manual inspection being extremely low throughput and susceptible to human bias, automatic counting software remains largely unused because of the wide range of variations in brightfield microscopy images. However, a robust automatic counting and cell classification solution would provide enormous benefits due to faster and more accurate quantitative results without human variability; researchers and medical professionals could better characterize stage-specific drug targets and better quantify patient reactions to drugs.

Previous attempts to automate the process of identifying and quantifying malaria have not gained major traction partly due to difficulty of replication, comparison, and extension. Authors also rarely make their image sets available, which precludes replication of results and assessment of potential improvements. The lack of a standard set of images nor standard set of metrics used to report results has impeded the field.</i>

Example images


0
200
400
600
800
1000
1200
0
250
500
750
1000
1250
1500
1750


0
50
100
150
200
250
0
50
100
150
200
250

Human Malaria					
Stages / Species	Ring	Trophozoite	Schizont	Gametocyte	
P. falciparum					• *Parasitised red cells (pRBCs) not enlarged.* • *RBCs containing mature trophozoites sequestered in deep vessels.* • *Total parasite biomass = circulating parasites + sequestered parasites.*
P. vivax					• *Parasites prefer young red cells* • *pRBCs enlarged.* • *Trophozoites are amoeboid in shape.* • *All stages present in peripheral blood.*
P. malariae					• *Parasites prefer old red cells.* • *pRBCs not enlarged.* • *Trophozoites tend to have a band shape.* • *All stages present in peripheral blood*
P. ovale					• *pRBCs slightly enlarged and have an oval shape, with tufted ends.* • *All stages present in peripheral blood.*
P. knowlesi					• *pRBCs not enlarged.* • *Trophozoites, pigment spreads inside cytoplasm, like P. malariae, band form may be seen* • *Multiple invasion & high parasitaemia can be seen like P. falciparum* • *All stages present in peripheral blood.*

Dataset :

Images :
Images are in .png or .jpg format. There are 3 sets of images consisting of 1364 images (~80,000 cells) with different researchers having prepared each one: from Brazil (Stefanie Lopes), from Southeast Asia (Benoit Malleret), and time course (Gabriel Rangel). Blood smears were stained with Giemsa reagent.
Website : https://data.broadinstitute.org/bbbc/BBBC041/ (https://data.broadinstitute.org/bbbc/BBBC041/)
</i>

Data Size : 2.26 GB
Data Source : https://data.broadinstitute.org/bbbc/BBBC041/malaria.zip (https://data.broadinstitute.org/bbbc/BBBC041/malaria.zip)

Ground truth

The data consists of two classes of uninfected cells (RBCs and leukocytes) and four classes of infected cells (gametocytes, rings, trophozoites, and schizonts). Annotators were permitted to mark some cells as difficult if not clearly in one of the cell classes. The data had a heavy imbalance towards uninfected RBCs versus uninfected leukocytes and infected cells, making up over 95% of all cells.
A class label and set of bounding box coordinates were given for each cell. For all data sets, infected cells were given a class label by Stefanie Lopes, malaria researcher at the Dr. Heitor Vieira Dourado Tropical Medicine Foundation hospital, indicating stage of development or marked as difficult.</i>

Cause of Malaria :

Although the malaria virus doesn't take the form of a mutant mosquito, it sure feels like a mutant problem. The deadly disease has reached epidemic, even endemic proportions in different parts of the world — killing around 400,000 people annually. In other areas of the world, it's virtually nonexistent. Some areas are just particularly prone to a disease outbreak — there are certain factors that make an area more likely to be infected by malaria.

1. ***High poverty levels***
2. ***Lack of access to proper healthcare***
3. ***Political instability***
4. ***Presence of disease transmission vectors (ex. mosquitos) </i>***

Current Diagnosing Methods :

The most widely used method (so far) is examining thin blood smears under a microscope, and visually searching for infected cells. The patients' blood is smeared on a glass slide and stained with contrasting agents to better identify infected parasites in their red blood cells. Then, a clinician manually counts the number of parasitic red blood cells — sometimes up to 5,000 cells.

Why a Deep Learning Model?

Deep Learning have the ability to automatically extract features and learn filters. In previous machine learning solutions, features had to be manually programmed in — for example, size, color, the morphology of the cells. Utilizing Deep Learning will greatly speed up prediction time while mirroring (or even exceeding) the accuracy of clinicians. I am going use Keras, with a Tensorflow backend to create models for Object Detection.

GIT Cloning Mask R-CNN module

In [0]:
```
!git clone https://github.com/matterport/Mask_RCNN
```

```
fatal: destination path 'Mask_RCNN' already exists and is not an empty direct
ory.
```

Navigating to Mask R-CNN Directory

In [0]:
```
import os
os.chdir('Mask_RCNN/')
```

Library Imports

In [0]:
```
from os import listdir
from numpy import *
from os import listdir
from mrcnn.utils import *
from mrcnn.visualize import *
from mrcnn.config import *
from mrcnn.model import *

from PIL import Image
import imageio
import warnings
```

```
import spacy
import os
warnings.filterwarnings("ignore")
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib import patches
import seaborn as sns
import numpy as np
from wordcloud import WordCloud
import datetime as dt
from sklearn import metrics
from sklearn.metrics import *
from datetime import datetime
import pickle
from collections import Counter
from scipy.sparse import hstack
from scipy.sparse import save_npz, load_npz
from sklearn.preprocessing import *
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from keras.models import *
from keras.layers import *
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.callbacks import *
from keras import *
from keras.optimizers import *
from prettytable import PrettyTable
from imutils import paths
import random
import shutil
from tqdm import tqdm
import json
import cv2
import tensorflow as tf
from scipy.io import loadmat
import shutil
import pprint
import sys
import time
from optparse import OptionParser
import math
import copy
from keras.engine.topology import get_source_inputs
from keras.utils import *
from keras.utils.data_utils import get_file
from tensorflow.keras.utils import Progbar
from keras.objectives import categorical_crossentropy
from keras.engine import *
from matplotlib.patches import Rectangle
import imgaug
```

```
The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade (https://www.tensorflow.org/guide/migrate) now or ensure
your notebook will continue to use TensorFlow 1.x via the %tensorflow_version 1.x
magic: more info
(https://colab.research.google.com/notebooks/tensorflow_version.ipynb).

Using TensorFlow backend.
```

In [0]:
```
## Run this if you are using Laptop or Dektop
#folderLocation = './Malaria Dataset/'
```

In [0]:
```
## Run this if you are using Google Colab

from google.colab import drive
drive.mount('/content/drive', force_remount=True)

folderLocation = '/content/drive/My Drive/Colab Notebooks/malaria/'
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client
_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&
redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=ema
il%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.
googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdri
ve.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.read
only

Enter your authorization code:
..........
Mounted at /content/drive
```

Extracting Data from train.json and test.json

In [0]:
```
start = datetime.now()

read_train_file = str(folderLocation)+'training.json'

with open(read_train_file) as file:
  train_data = json.load(file)

read_test_file = str(folderLocation)+'test.json'

with open(read_test_file) as file:
  test_data = json.load(file)

print("\n\nTime taken to run this cell :", datetime.now() - start)
```

```
Time taken to run this cell : 0:00:02.347353
```

Creating Train Dataframe

In [0]:
```
start = datetime.now()

train_file_name = str(folderLocation)+'train_dataframe.csv'

if os.path.exists(train_file_name):
  print('Data is already Created No Need to Create Again, Loading Data : ')
  train_dataframe = pd.read_csv(train_file_name,header=0)
else:
  print('Data is Not Created, Creating Data : ')
  image_name=''
  image_url=''
  xmin=0
  xmax=0
  ymin=0
  ymax=0
  category=''

  rowsList = list()

  for list_item1 in train_data:
    for key1, value1 in list_item1.items():
      if key1 == 'image':
        for key2, value2 in value1.items():
          if key2 != 'shape':
            image = value2[8:]
            image_url = value2
      elif key1 == 'objects':
        for list_item2 in value1:
          for key2, value2 in list_item2.items():
            if key2 == 'bounding_box':
              columnList = list()
              for key3,value3 in value2.items():
                for key4, value4 in value3.items():
                  if key3=='minimum':
                    xmin = value3['r']
                    ymin = value3['c']
                  elif key3=='maximum':
                    xmax = value3['r']
                    ymax = value3['c']
            else:
              if key2 == 'category':
                category = value2
                columnList=[image,image_url,xmin,ymin,xmax,ymax,category]
                rowsList.append(columnList)
  train_dataframe = pd.DataFrame(rowsList,columns=['image_name','image_url','x
min','ymin','xmax','ymax','category'])
  train_dataframe.replace(to_replace='red blood cell', value='rbc',inplace=Tru
e)
  train_dataframe['image_id'] = train_dataframe.groupby(['image_name']).ngroup
()
  train_dataframe.sort_values(by=['image_id'], inplace=True)
  train_dataframe = train_dataframe[['image_id','image_name','image_url','xmi
n','ymin','xmax','ymax','category']]
  #train_dataframe.set_index('image_id', inplace=True)
  train_dataframe.to_csv(train_file_name,index=False)

print("\n\nTime taken to run this cell :", datetime.now() - start)
```

```
Data is already Created No Need to Create Again, Loading Data :


Time taken to run this cell : 0:00:00.615572
```

In [0]:
```
train_dataframe.head()
```

Out[0]:

	image_id	image_name	image_url	xmin	ymin	xmax	ymax	category
0	0	002f20ad-2ace-499c-9335-c9080bc3e6b5.png	/images/002f20ad-2ace-499c-9335-c9080bc3e6b5.png	926	52	1048	173	rbc
1	0	002f20ad-2ace-499c-9335-c9080bc3e6b5.png	/images/002f20ad-2ace-499c-9335-c9080bc3e6b5.png	1013	1384	1115	1514	rbc
2	0	002f20ad-2ace-499c-9335-c9080bc3e6b5.png	/images/002f20ad-2ace-499c-9335-c9080bc3e6b5.png	37	1459	153	1590	rbc
3	0	002f20ad-2ace-499c-9335-c9080bc3e6b5.png	/images/002f20ad-2ace-499c-9335-c9080bc3e6b5.png	556	1335	665	1445	rbc
4	0	002f20ad-2ace-499c-9335-c9080bc3e6b5.png	/images/002f20ad-2ace-499c-9335-c9080bc3e6b5.png	824	998	941	1115	rbc

In [0]:
```
print('Train Data Shape : ',train_dataframe.shape)
```

```
Train Data Shape :  (80113, 8)
```

Creating Test Dataframe

In [0]:
```
start = datetime.now()

test_file_name = str(folderLocation)+'test_dataframe.csv'

if os.path.exists(test_file_name):
  print('Data is already Created No Need to Create Again, Loading Data : ')
  test_dataframe = pd.read_csv(test_file_name,header=0)
else:
  print('Data is Not Created, Creating Data : ')
  image_name=''
  image_url=''
  xmin=0
  xmax=0
  ymin=0
  ymax=0
  category=''

  rowsList = list()

  for list_item1 in test_data:
    for key1, value1 in list_item1.items():
      if key1 == 'image':
        for key2, value2 in value1.items():
          if key2 != 'shape':
            image = value2[8:]
            image_url = value2
      elif key1 == 'objects':
        for list_item2 in value1:
          for key2, value2 in list_item2.items():
            if key2 == 'bounding_box':
              columnList = list()
              for key3,value3 in value2.items():
                for key4, value4 in value3.items():
                  if key3=='minimum':
                    xmin = value3['r']
                    ymin = value3['c']
                  elif key3=='maximum':
                    xmax = value3['r']
                    ymax = value3['c']
            else:
              if key2 == 'category':
                category = value2
                columnList=[image,image_url,xmin,ymin,xmax,ymax,category]
                rowsList.append(columnList)
  test_dataframe = pd.DataFrame(rowsList,columns=['image_name','image_url','xmin','ymin','xmax','ymax','category'])
  test_dataframe.replace(to_replace='red blood cell', value='rbc',inplace=True)
  test_dataframe['image_id'] = test_dataframe.groupby(['image_name']).ngroup()
  test_dataframe.sort_values(by=['image_id'], inplace=True)
  test_dataframe = test_dataframe[['image_id','image_name','image_url','xmin','ymin','xmax','ymax','category']]
  #test_dataframe.set_index('image_id', inplace=True)
  test_dataframe.to_csv(test_file_name,index=False)

print("\n\nTime taken to run this cell :", datetime.now() - start)
```

```
Data is already Created No Need to Create Again, Loading Data :


Time taken to run this cell : 0:00:00.382571
```

In [0]:
```
test_dataframe.head()
```

Out[0]:

	image_id	image_name	image_url	xmin	ymin	xmax	ymax	category
0	0	010961af-b38c-49de-aca0-e3732d73d414.jpg	/images/010961af-b38c-49de-aca0-e3732d73d414.jpg	0	923	144	1060	rbc
1	0	010961af-b38c-49de-aca0-e3732d73d414.jpg	/images/010961af-b38c-49de-aca0-e3732d73d414.jpg	147	674	307	819	rbc
2	0	010961af-b38c-49de-aca0-e3732d73d414.jpg	/images/010961af-b38c-49de-aca0-e3732d73d414.jpg	1245	1497	1363	1602	rbc
3	0	010961af-b38c-49de-aca0-e3732d73d414.jpg	/images/010961af-b38c-49de-aca0-e3732d73d414.jpg	1065	624	1173	716	rbc
4	0	010961af-b38c-49de-aca0-e3732d73d414.jpg	/images/010961af-b38c-49de-aca0-e3732d73d414.jpg	617	1472	739	1598	rbc

In [0]:
```
print('Test Data Shape : ',test_dataframe.shape)
```

```
Test Data Shape :  (5922, 8)
```

Total Number of Images in Train

In [0]:
```
# Number of unique training images
train_dataframe['image_name'].nunique()
```

Out[0]:
```
1208
```

Bounding Boxes Per Images in Train

In [0]:
```
# Number of unique training images
train_dataframe['image_name'].value_counts()
```

Out[0]:
```
252071ac-e9f2-41d2-abbb-87ef01b8a997.png    223
fe851c88-692d-4199-87e0-d19d9c4eb591.png    219
947348fe-4c70-468e-8009-3c14daeca69b.png    217
9f38055f-4fe6-47fd-a6dd-c0743db4eccf.png    216
b10ea12f-cd36-4f2c-994a-67d1665269ab.png    203
                                           ...
0403835d-43f4-4a97-abfc-b2ff915f6d7b.png     13
98efca97-37d2-4459-aca6-375ecc0c4539.png     12
89b276cc-cc8f-4378-a877-e01aff333373.png     11
361a3578-4cbb-422b-9e3c-b5e61395b12e.png      9
bbf687b5-c6f9-4821-b2e5-a25df1acba47.png      9
Name: image_name, Length: 1208, dtype: int64
```

Total Number of Images in Test

In [0]:
```
# Number of unique test images
test_dataframe['image_name'].nunique()
```

Out[0]:
```
120
```

Bounding Boxes Per Images in Test

In [0]:
```
# Number of unique test images
test_dataframe['image_name'].value_counts()
```

Out[0]:
```
511b6e13-acd7-4dd3-b1d0-c4aa10a514c7.jpg    108
f0ee03d6-c57b-43a0-8812-359330bdb93a.jpg    104
b9025f55-789d-4acb-9c0a-34e877c95c5c.jpg     98
8448ac8c-fa7a-475a-9ca2-dc0174f78a39.jpg     89
294bc238-efdb-4f88-b83b-f41fe5750106.jpg     85
                                           ...
0e0bc36f-6fb2-4763-b0a1-b4c67b2d2d2a.jpg     22
7d16d1e8-4abf-46fc-81bc-d71bd08ed6f2.jpg     22
54c88d45-30c3-46f8-8fef-b3e9c07781f2.jpg     14
bd4da4f8-8996-47e0-a4e5-a3da5717441e.jpg     13
ebb53337-1d4f-4ab5-9b6d-83a66f48fe2e.jpg     12
Name: image_name, Length: 120, dtype: int64
```

In [0]:
```
type(test_dataframe['image_name'].value_counts())
```

Out[0]:
```
pandas.core.series.Series
```

Total Number of Distinct Classes and data points for each

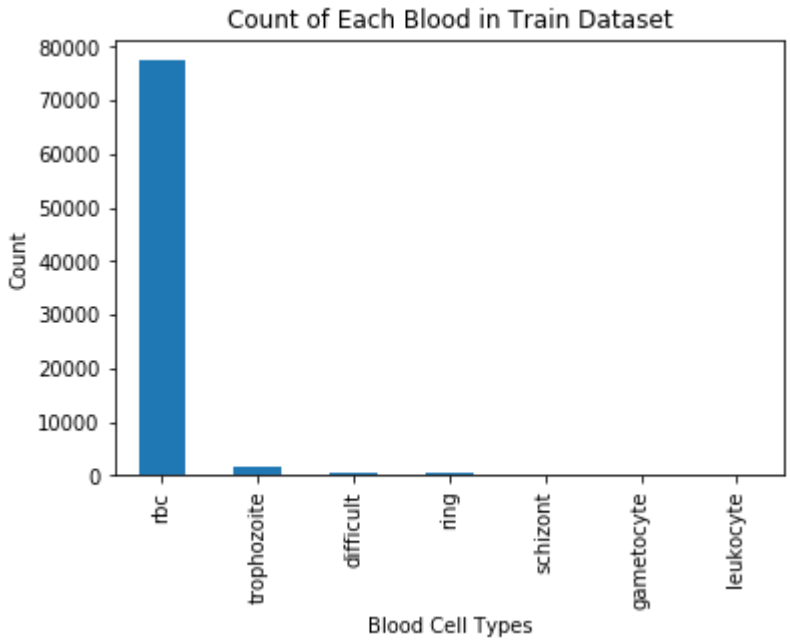
In [0]:
```
# Number of classes
train_dataframe['category'].value_counts()
```

Out[0]:
```
rbc            77420
trophozoite     1473
difficult        441
ring             353
schizont         179
gametocyte       144
leukocyte        103
Name: category, dtype: int64
```

Category Plot

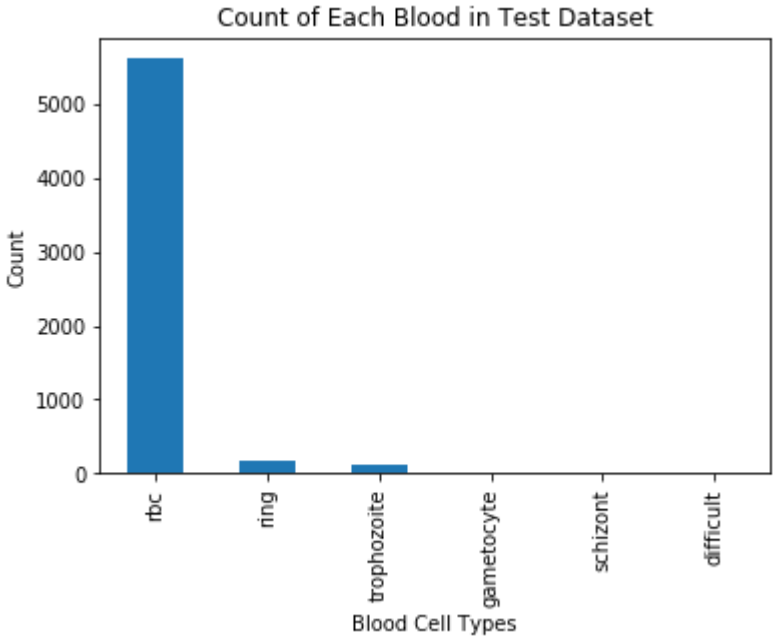
In [0]:
```
df = train_dataframe
pd.value_counts(df['category']).plot(kind="bar")
plt.xlabel("Blood Cell Types")
plt.ylabel("Count")
plt.title("Count of Each Blood in Train Dataset")
```

Out[0]:
```
Text(0.5, 1.0, 'Count of Each Blood in Train Dataset')
```



In [0]:
```
df = test_dataframe
pd.value_counts(df['category']).plot(kind="bar")
plt.xlabel("Blood Cell Types")
plt.ylabel("Count")
plt.title("Count of Each Blood in Test Dataset")
```

Out[0]: Text(0.5, 1.0, 'Count of Each Blood in Test Dataset')



Conclusion :

1. As we can see 'rbc' category has the most points, clearly dominating the dataset.
2. 'ring' has some points but not even close to 'rbc'
3. 'trophozoite' has lesser points than 'ring'
4. Other categories as so less that cannot be even seen in the graph

Creating a Temp Dataframe

In [0]:
```
start = datetime.now()

temp_file_name = str(folderLocation)+'temp_dataframe.csv'

if os.path.exists(temp_file_name):
    print('Data already created.. Hence Loading.')
    data_concat = pd.read_csv(temp_file_name,header=0)
else:
    print('Creating Data...')
    train_temp = train_dataframe[['image_name','xmin','ymin','xmax','ymax','ca
tegory']].copy()
    test_temp = test_dataframe[['image_name','xmin','ymin','xmax','ymax','cate
gory']].copy()
    train_temp['isTrain'] = 1
    test_temp['isTrain'] = 0
    if 'image_id' in train_temp.columns:
        train_temp.drop(columns=['image_id'], inplace=True)
    if 'image_id' in test_temp.columns:
        test_temp.drop(columns=['image_id'], inplace=True)
    frames = [train_temp,test_temp]
    data_concat = pd.concat(frames,ignore_index=True)
    data_concat['image_id'] = data_concat.groupby(['image_name']).ngroup()
    data_concat.sort_values(by=['image_id'], inplace=True)
    #data_concat.set_index('image_id', inplace=True)
    data_concat = data_concat[['image_id','image_name','image_url','xmin','ymi
n','xmax','ymax','category','isTrain']]
    data_concat.to_csv(temp_file_name,index=False)
    print('Data Creation Done..')

print("\n\nTime taken to run this cell :", datetime.now() - start)
```

```
Data already created.. Hence Loading.


Time taken to run this cell : 0:00:01.798191
```

In [0]:
```
data_concat.shape
```

Out[0]:
```
(86035, 9)
```

In [0]:
```
data_concat.head()
```

Out[0]:

	image_id	image_name	image_url	xmin	ymin	xmax	ymax	category	isTrain
0	0	002f20ad-2ace-499c-9335-c9080bc3e6b5.png	/images/002f20ad-2ace-499c-9335-c9080bc3e6b5.png	926	52	1048	173	rbc	1
1	0	002f20ad-2ace-499c-9335-c9080bc3e6b5.png	/images/002f20ad-2ace-499c-9335-c9080bc3e6b5.png	338	465	471	600	rbc	1
2	0	002f20ad-2ace-499c-9335-c9080bc3e6b5.png	/images/002f20ad-2ace-499c-9335-c9080bc3e6b5.png	225	1056	329	1169	rbc	1
3	0	002f20ad-2ace-499c-9335-c9080bc3e6b5.png	/images/002f20ad-2ace-499c-9335-c9080bc3e6b5.png	982	1254	1092	1365	rbc	1
4	0	002f20ad-2ace-499c-9335-c9080bc3e6b5.png	/images/002f20ad-2ace-499c-9335-c9080bc3e6b5.png	609	1072	733	1195	rbc	1

Defining classes as List

In [0]:
```
CLASSES = ['rbc','trophozoite','difficult','ring','schizont','gametocyte','leukocyte']
num_classes_category = len(CLASSES)
```

Defining Malaria Dataset

In [0]:
```
image_path_dict = dict()

# class that defines and loads the Malaria dataset
class Malaria(Dataset):
    # load the dataset definitions
    def load_dataset(self,is_train=True):

        if is_train:
            dataset = data_concat[data_concat.isTrain == 1].groupby(['image_id','image_name'])
            images_dir = folderLocation + 'images/'
            #dataset = data_concat[data_concat.image_id < 1100].groupby(['image_id','image_name'])
        else:
            dataset = data_concat[data_concat.isTrain == 0].groupby(['image_id','image_name'])
            images_dir = folderLocation + 'images/'
            #dataset = data_concat[data_concat.image_id >= 1100].groupby(['image_id','image_name'])

        self.add_class('malaria', 1, 'rbc')
        self.add_class('malaria', 2, 'trophozoite')
```

```
        self.add_class('malaria', 3, 'difficult')
        self.add_class('malaria', 4, 'ring')
        self.add_class('malaria', 5, 'schizont')
        self.add_class('malaria', 6, 'gametocyte')
        self.add_class('malaria', 7, 'leukocyte')

        # find all images
        for row,_ in dataset.size().iteritems():
            # extract image id
            image_id = row[0]
            img_path = images_dir + row[1]
            # add to dataset
            image_path_dict[image_id] = img_path
            self.add_image('malaria', image_id=image_id, path=img_path)

    def load_image(self, image_id):
        images_dir = folderLocation + 'images/'
        dataset = data_concat

        #filePath = self.image_reference(image_id)
        filePath = image_path_dict[image_id]
        #print('\n Image Loading From : ',filePath)

        # Load image
        img = Image.open( open(filePath, "rb") )
        img.load()
        image = np.asarray( img, dtype="int32" )
        #image = skimage.io.imread(filePath)
        #image = cv2.imread(filePath)
        #image = imageio.imread(filePath)
        #image = plt.imread(filePath)
        #image = Image.open(filePath)

        # If grayscale. Convert to RGB for consistency.
        if image.ndim != 3:
            image = skimage.color.gray2rgb(image)
        # If has an alpha channel, remove it for consistency
        if image.shape[-1] == 4:
            image = image[..., :3]
        return image

    # extract bounding boxes from an annotation file
    def extract_boxes(self, image_id):
        # extract each bounding box
        boxes = list()
        classes = list()
        dataset = data_concat
        #print()
        for _,row in dataset[dataset.image_id == image_id].iterrows():
            xmin = row.xmin
            xmax = row.xmax
            ymin = row.ymin
            ymax = row.ymax
            class_name = row.category
            #coors = [xmin, ymin,  xmax, ymax]
            coors = [ymin, xmin, ymax, xmax]
            boxes.append(coors)
```

```
            classes.append(class_name)
        return boxes, classes

    # load the masks for an image
    def load_mask(self, image_id):
        # get details of image
        #info = self.image_info[image_id]

        image = self.load_image(image_id)
        # load XML
        boxes, classes = self.extract_boxes(image_id)
        # create one array for all masks, each on a different channel
        masks = zeros([image.shape[0], image.shape[1], len(boxes)], dtype='uin
t8')
        # create masks
        class_ids = list()
        for i in range(len(boxes)):
            box = boxes[i]
            row_s, row_e = box[1], box[3]
            col_s, col_e = box[0], box[2]
            masks[row_s:row_e, col_s:col_e, i] = 1
            class_ids.append(self.class_names.index(classes[i]))
        return masks, np.asarray(class_ids)

    # load an image reference
    def image_reference(self, image_id):
        path = ''
        for image_dict in self.image_info:
            if image_dict['id'] == image_id:
                path = image_dict['path']
                break;
        return path
```

Defining Infected Dataset

In [0]:
```
# class that defines and loads the Malaria dataset
class Infected(Dataset):
    # load the dataset definitions
    def load_dataset(self,is_train=True):

        if is_train:
            dataset = data_concat[data_concat.isTrain == 1].groupby(['image_id','image_name'])
            images_dir = folderLocation + 'images/'
            #dataset = data_concat[data_concat.image_id < 1100].groupby(['image_id','image_name'])
        else:
            dataset = data_concat[data_concat.isTrain == 0].groupby(['image_id','image_name'])
            images_dir = folderLocation + 'images/'
            #dataset = data_concat[data_concat.image_id >= 1100].groupby(['image_id','image_name'])

        self.add_class('infected', 1, 'rbc')
        self.add_class('infected', 2, 'trophozoite')
```

```
        self.add_class('infected', 3, 'difficult')
        self.add_class('infected', 4, 'ring')
        self.add_class('infected', 5, 'schizont')
        self.add_class('infected', 6, 'gametocyte')
        self.add_class('infected', 7, 'leukocyte')


        # find all images
        for row,_ in dataset.size().iteritems():
            # extract image id
            image_id = row[0]
            img_path = images_dir + row[1]
            # add to dataset
            image_path_dict[image_id] = img_path
            self.add_image('infected', image_id=image_id, path=img_path)

    def load_image(self, image_id):
        images_dir = folderLocation + 'images/'
        dataset = data_concat

        #filePath = self.image_reference(image_id)
        filePath = image_path_dict[image_id]
        #print('\n Image Loading From : ',filePath)

        # Load image
        img = Image.open( open(filePath, "rb") )
        img.load()
        image = np.asarray( img, dtype="int32" )
        #image = skimage.io.imread(filePath)
        #image = cv2.imread(filePath)
        #image = imageio.imread(filePath)
        #image = plt.imread(filePath)
        #image = Image.open(filePath)

        # If grayscale. Convert to RGB for consistency.
        if image.ndim != 3:
            image = skimage.color.gray2rgb(image)
        # If has an alpha channel, remove it for consistency
        if image.shape[-1] == 4:
            image = image[..., :3]
        return image

    # extract bounding boxes from an annotation file
    def extract_boxes(self, image_id):
        # extract each bounding box
        boxes = list()
        classes = list()
        dataset = data_concat
        #print()
        for _,row in dataset[dataset.image_id == image_id].iterrows():
            if row.category == 'rbc':
              continue
            xmin = row.xmin
            xmax = row.xmax
            ymin = row.ymin
            ymax = row.ymax
            class_name = row.category
```

```
            #coors = [xmin, ymin,  xmax, ymax]
            coors = [ymin, xmin, ymax, xmax]
            boxes.append(coors)
            classes.append(class_name)
        return boxes, classes

    # load the masks for an image
    def load_mask(self, image_id):
        # get details of image
        #info = self.image_info[image_id]

        image = self.load_image(image_id)
        # load XML
        boxes, classes = self.extract_boxes(image_id)
        # create one array for all masks, each on a different channel
        masks = zeros([image.shape[0], image.shape[1], len(boxes)], dtype='uin
t8')
        # create masks
        class_ids = list()
        for i in range(len(boxes)):
            box = boxes[i]
            row_s, row_e = box[1], box[3]
            col_s, col_e = box[0], box[2]
            masks[row_s:row_e, col_s:col_e, i] = 1
            class_ids.append(self.class_names.index(classes[i]))
        return masks, np.asarray(class_ids)

    # load an image reference
    def image_reference(self, image_id):
        path = ''
        for image_dict in self.image_info:
            if image_dict['id'] == image_id:
                path = image_dict['path']
                break;
        return path
```

Creating Train & Test Dataset

In [0]:
```
start = datetime.now()

# train set
train_set = Malaria()
train_set.load_dataset(is_train=True)
train_set.prepare()
print('Train: %d' % len(train_set.image_ids))

test_set = Malaria()
test_set.load_dataset(is_train=False)
test_set.prepare()
print('Test: %d' % len(test_set.image_ids))

print("\n\nTime taken to run this cell :", datetime.now() - start)
```

```
Train: 1208
Test: 120


Time taken to run this cell : 0:00:00.023708
```

Creating Infected Train & Test Data set

In [0]:
```
start = datetime.now()

# train set
train_set_infected = Infected()
train_set_infected.load_dataset(is_train=True)
train_set_infected.prepare()
print('Train: %d' % len(train_set_infected.image_ids))

test_set_infected = Infected()
test_set_infected.load_dataset(is_train=False)
test_set_infected.prepare()
print('Test: %d' % len(test_set_infected.image_ids))

print("\n\nTime taken to run this cell :", datetime.now() - start)
```

```
Train: 1208
Test: 120


Time taken to run this cell : 0:00:00.022725
```

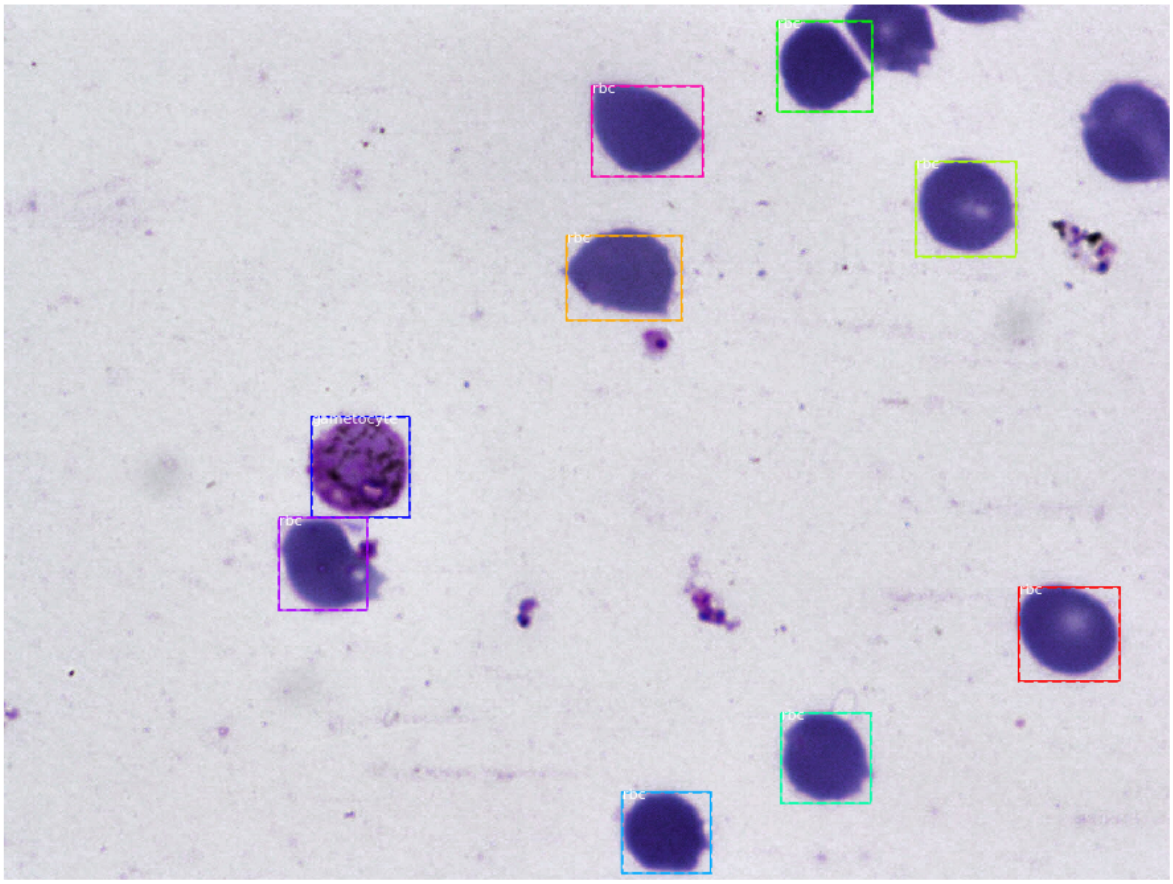
Test Malaria Dataset Object

In [0]:
```
image_name = '2afe2750-c963-46f9-a1ad-9260e80679ca.png'
data_concat[data_concat.image_name==image_name].head()
```

Out[0]:

	image_id	image_name	image_url	xmin	ymin	xmax	ymax	category	isTr
14175	216	2afe2750-c963-46f9-a1ad-9260e80679ca.png	/images/2afe2750-c963-46f9-a1ad-9260e80679ca.png	66	1279	169	1394	rbc	
14176	216	2afe2750-c963-46f9-a1ad-9260e80679ca.png	/images/2afe2750-c963-46f9-a1ad-9260e80679ca.png	308	893	385	1056	rbc	
14177	216	2afe2750-c963-46f9-a1ad-9260e80679ca.png	/images/2afe2750-c963-46f9-a1ad-9260e80679ca.png	887	1059	1012	1194	rbc	
14178	216	2afe2750-c963-46f9-a1ad-9260e80679ca.png	/images/2afe2750-c963-46f9-a1ad-9260e80679ca.png	480	599	585	711	rbc	
14179	216	2afe2750-c963-46f9-a1ad-9260e80679ca.png	/images/2afe2750-c963-46f9-a1ad-9260e80679ca.png	435	1334	542	1449	rbc	

In [0]:
```
start = datetime.now()

# define image id
image_id = 974
# load the image
image = train_set.load_image(image_id)
# load the masks and the class ids
mask, class_ids = train_set.load_mask(image_id)
# extract bounding boxes from the masks
bbox = extract_bboxes(mask)
# display image with masks and bounding boxes
display_instances(image, bbox, mask, class_ids, train_set.class_names, show_ma
sk=False)

print("\n\nTime taken to run this cell :", datetime.now() - start)
```



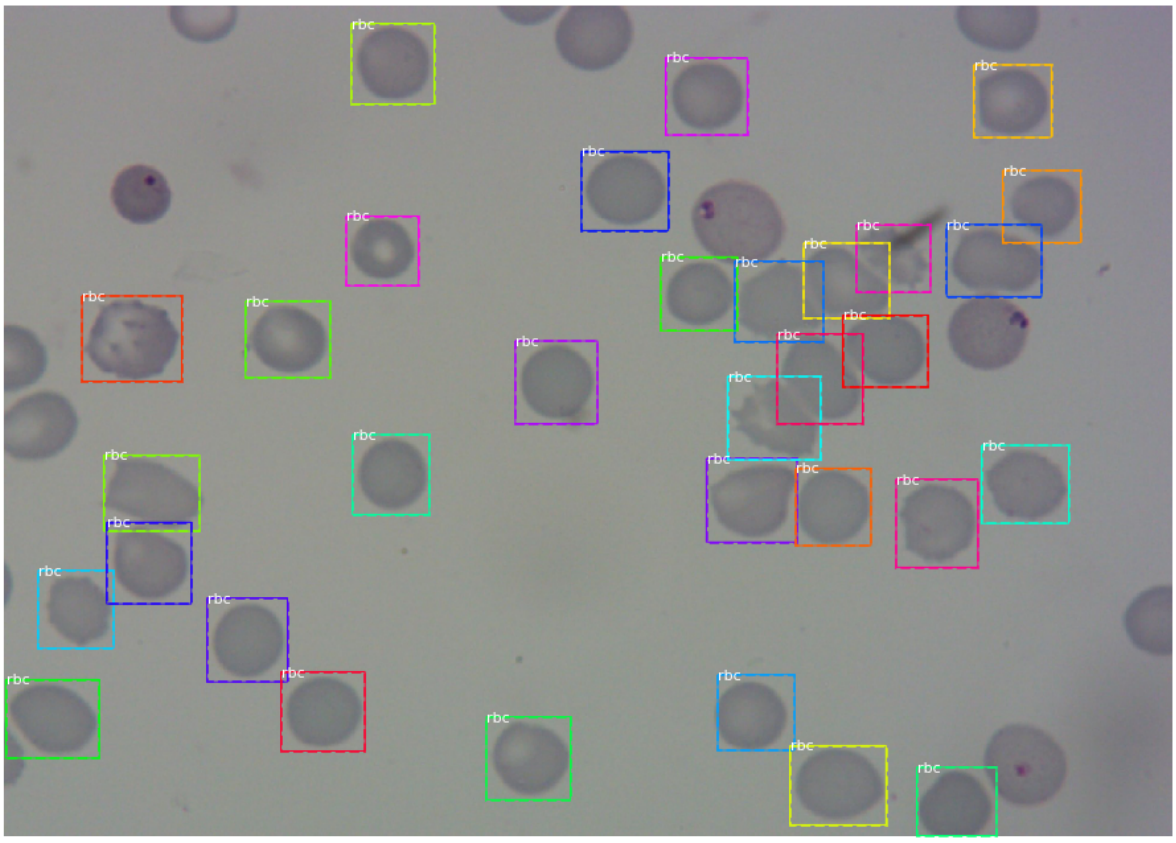
Time taken to run this cell : 0:00:01.736373

In [0]:
```
image_name = 'c328260d-bafa-4aaa-9e7a-79bc165d7fdb.jpg'
data_concat[data_concat.image_name==image_name].head()
```

Out[0]:

	image_id	image_name	image_url	xmin	ymin	xmax	ymax	category	isTr
65727	1013	c328260d-bafa-4aaa-9e7a-79bc165d7fdb.jpg	/images/c328260d-bafa-4aaa-9e7a-79bc165d7fdb.jpg	365	1420	477	1543	rbc	
65728	1013	c328260d-bafa-4aaa-9e7a-79bc165d7fdb.jpg	/images/c328260d-bafa-4aaa-9e7a-79bc165d7fdb.jpg	395	1332	520	1475	rbc	
65729	1013	c328260d-bafa-4aaa-9e7a-79bc165d7fdb.jpg	/images/c328260d-bafa-4aaa-9e7a-79bc165d7fdb.jpg	940	58	1070	184	rbc	
65730	1013	c328260d-bafa-4aaa-9e7a-79bc165d7fdb.jpg	/images/c328260d-bafa-4aaa-9e7a-79bc165d7fdb.jpg	1114	1189	1240	1317	rbc	
65731	1013	c328260d-bafa-4aaa-9e7a-79bc165d7fdb.jpg	/images/c328260d-bafa-4aaa-9e7a-79bc165d7fdb.jpg	350	571	466	691	rbc	

In [0]:
```
start = datetime.now()

# define image id
image_id = 1013
# load the image
image = test_set.load_image(image_id)
# load the masks and the class ids
mask, class_ids = test_set.load_mask(image_id)
# extract bounding boxes from the masks
bbox = extract_bboxes(mask)
# display image with masks and bounding boxes
display_instances(image, bbox, mask, class_ids, train_set.class_names, show_mask=False)

print("\n\nTime taken to run this cell :", datetime.now() - start)
```



Time taken to run this cell : 0:00:03.728339

Defining Class_weight

In [0]:
```
def create_class_weight(labels_dict,mu=0.15):
    total = np.sum(list(labels_count.values()))
    keys = labels_dict.keys()
    class_weight = dict()

    for key in keys:
        score = math.log(mu*total/float(labels_dict[key]))
        class_weight[key] = score if score > 1.0 else 1.0

    return class_weight
```

In [0]:
```
labels_count = dict(train_dataframe['category'].value_counts())
print(labels_count)
class_weight = create_class_weight(labels_count)
print(class_weight)
```

```
{'rbc': 77420, 'trophozoite': 1473, 'difficult': 441, 'ring': 353, 'schizon
t': 179, 'gametocyte': 144, 'leukocyte': 103}
{'rbc': 1.0, 'trophozoite': 2.099017015667774, 'difficult': 3.305028556683558
4, 'ring': 3.5276053751971075, 'schizont': 4.206687626289649, 'gametocyte':
4.424260132554403, 'leukocyte': 4.759344443900768}
```

Train Mask R-CNN Model for Malaria Detection & Infection Detection

Defining Configuration Class for training

In [0]:
```
# define a configuration for the model
class MalariaConfig(Config):
    # Give the configuration a recognizable name
    NAME = "malaria_cfg"
    # Number of classes (background + cell)
    NUM_CLASSES = num_classes_category + 1
    # Number of training steps per epoch
    STEPS_PER_EPOCH = 1000
    VALIDATION_STEPS = 500
    IMAGES_PER_GPU = 1
    IMAGE_MIN_DIM = 512
    IMAGE_MAX_DIM = 512

# prepare config
config = MalariaConfig()
```

In [0]:
```
# define a configuration for the model
class InfectedConfig(Config):
    # Give the configuration a recognizable name
    NAME = "infected_cfg"
    # Number of classes (background + cell)
    NUM_CLASSES = num_classes_category + 1
    # Number of training steps per epoch
    STEPS_PER_EPOCH = 1000
    VALIDATION_STEPS = 500
    IMAGES_PER_GPU = 1
    IMAGE_MIN_DIM = 512
    IMAGE_MAX_DIM = 512

# prepare config
infectedConfig = InfectedConfig()
```

Model Directory Path

In [0]:
```
model_directory = folderLocation + 'model/'
```

Defining Models :

In [0]:
```
# define the model
model = MaskRCNN(mode='training', model_dir=model_directory, config=config)
```

In [0]:
```
# define the model
model_infected = MaskRCNN(mode='training', model_dir=model_directory, config=i
nfectedConfig)
```

Loading Base Weights :

In [0]:
```
base_weight = model.find_last()
print(base_weight)
# load weights (mscoco)
model.load_weights(base_weight, by_name=True, exclude=["mrcnn_class_logits",
"mrcnn_bbox_fc",  "mrcnn_bbox", "mrcnn_mask"])
```

In [0]:
```
base_weight_infected = model_infected.find_last()
print(base_weight_infected)
# load weights (mscoco)
model_infected.load_weights(base_weight_infected, by_name=True, exclude=["mrcn
n_class_logits", "mrcnn_bbox_fc",  "mrcnn_bbox", "mrcnn_mask"])
```

Training the Models :

In [0]:
```
# train weights (output layers or 'heads')
model.train(train_dataset=train_set, val_dataset=test_set, learning_rate=confi
g.LEARNING_RATE, epochs=50, layers='heads')
```

In [0]:
```
# train weights (output layers or 'heads')
model_infected.train(train_dataset=train_set_infected, val_dataset=test_set_in
fected, learning_rate=infectedConfig.LEARNING_RATE, epochs=20, layers='heads')
```

Evaluate a Mask R-CNN Model

Defining Configuration Class for Testing

In [0]:
```
# define the prediction configuration
class PredictionConfig(Config):
        # define the name of the configuration
        NAME = "malaria_cfg"
        # number of classes (background + cell)
        NUM_CLASSES = num_classes_category + 1
        # simplify GPU config
        GPU_COUNT = 1
        IMAGES_PER_GPU = 1
        IMAGE_MIN_DIM = 512
        IMAGE_MAX_DIM = 512
        STEPS_PER_EPOCH = 500
```

In [0]:
```
# define the prediction configuration
class InfectedPredictionConfig(Config):
        # define the name of the configuration
        NAME = "infected_cfg"
        # number of classes (background + cell - rbc)
        NUM_CLASSES = num_classes_category + 1
        # simplify GPU config
        GPU_COUNT = 1
        IMAGES_PER_GPU = 1
        IMAGE_MIN_DIM = 512
        IMAGE_MAX_DIM = 512
        STEPS_PER_EPOCH = 500
```

Defining Model Directory Path

In [0]:
```
model_directory = folderLocation + 'model/'
```

Creating Models for Prediction

In [ ]:
```
# create config
cfg = PredictionConfig()
# define the model
model = MaskRCNN(mode='inference', model_dir=model_directory, config=cfg)
```

In [0]:
```
# create config
infectedCfg = InfectedPredictionConfig()
# define the model
model_infected = MaskRCNN(mode='inference', model_dir=model_directory, config=
infectedCfg)
```

Loading the Trained weights

In [ ]:
```
trained_weight = model.find_last()
# load model weights
model.load_weights(trained_weight, by_name=True)
```

In [0]:
```
trained_weight_infected = model_infected.find_last()
# load model weights
model_infected.load_weights(trained_weight_infected, by_name=True)
```

```
Re-starting from epoch 20
```

Evaluating Model : Calculating Mean Average Precision (mAP)

In [0]:
```
# calculate the mAP for a model on a given dataset
def evaluate_model(dataset, model, cfg, iou_threshold=0.5):
        APs = list()
        for image_id in dataset.image_ids:
                # load image, bounding boxes and masks for the image id
                image, image_meta, gt_class_id, gt_bbox, gt_mask = load_image_
gt(dataset, cfg, image_id, use_mini_mask=False)
                # convert pixel values (e.g. center)
                scaled_image = mold_image(image, cfg)
                # convert image into one sample
                sample = expand_dims(scaled_image, 0)
                # make prediction
                yhat = model.detect(sample, verbose=0)
                # extract results for first sample
                r = yhat[0]
                # calculate statistics, including AP
                AP, _, _, _ = compute_ap(gt_bbox, gt_class_id, gt_mask, r["roi
s"], r["class_ids"], r["scores"], r['masks'],iou_threshold=iou_threshold)
                # store
                APs.append(AP)
        # calculate the mean AP across all images
        mAP = mean(APs)
        return mAP
```

```
# calculate the mAP for a model on a given dataset
def evaluate_model_infected(dataset, model, cfg, iou_threshold=0.5):
        APs = list()
        for image_id in dataset.image_ids:
                original_image, image_meta, gt_class_id, gt_bbox, gt_mask =loa
d_image_gt(dataset, cfg, image_id, use_mini_mask=False)
                results = model_infected.detect([original_image], verbose=0)
                r = results[0]
                # calculate statistics, including AP
                AP, _, _, _ = compute_ap(gt_bbox, gt_class_id, gt_mask, r["roi
s"], r["class_ids"], r["scores"], r['masks'],iou_threshold=iou_threshold)
                # store
                if not math.isnan(AP) and AP > 0.0:
                        APs.append(AP)
        # calculate the mean AP across all images
        mAP = mean(APs)
        return mAP

def evaluate_both_models(iou_threshold=0.5):
  APs = list()
  for image_id in test_set.image_ids:
    # load image, bounding boxes and masks for the image id
    image, _, gt_class_id, gt_bbox, gt_mask = load_image_gt(test_set, cfg, ima
ge_id, use_mini_mask=False)
    # convert pixel values (e.g. center)
    scaled_image = mold_image(image, cfg)
    # convert image into one sample
    sample = expand_dims(scaled_image, 0)
    # make prediction
    yhat = model.detect(sample, verbose=0)
    # extract results for first sample
    r = yhat[0]

    original_image, _, gt_class_id_inf, gt_bbox_inf, gt_mask_inf =load_image_g
t(test_set_infected, infectedCfg, image_id, use_mini_mask=False)
    results = model_infected.detect([original_image], verbose=0)
    r2 = results[0]

    roisr1r2 = np.concatenate((r['rois'],r2['rois']))
    try:
      masksr1r2 = tf.concat((r['masks'],r2['masks']),axis=2).eval(session=tf.c
ompat.v1.Session())
    except:
      masksr1r2 = r['masks']
    classIdsr1r2 = np.concatenate((r['class_ids'],r2['class_ids']))
    scoresr1r2 = np.concatenate((r['scores'],r2['scores']))
    # calculate statistics, including AP
    AP, _, _, _ = compute_ap(gt_bbox, gt_class_id, gt_mask, r["rois"], r["clas
s_ids"], r["scores"], r['masks'],iou_threshold=iou_threshold)
    # store
    APs.append(AP)
          # calculate the mean AP across all images
  mAP = mean(APs)
  return mAP
```

Evaluate Model 1 :

In [0]:
```
start = datetime.now()

# evaluate model on test dataset
train_mAP = evaluate_model(train_set, model, cfg)
print("Train mAP: %.3f" % train_mAP)

# evaluate model on test dataset
test_mAP = evaluate_model(test_set, model, cfg)
print("Test mAP: %.3f" % test_mAP)

print("\n\nTime taken to run this cell :", datetime.now() - start)
```

```
Train mAP: 0.832
Test mAP: 0.853


Time taken to run this cell : 0:27:46.913972
```

Evaluate Model 2 :

In [0]:
```
start = datetime.now()

# evaluate model on test dataset
train_mAP = evaluate_model_infected(train_set_infected, model_infected, infect
edCfg)
print("Train mAP: %.3f" % train_mAP)

# evaluate model on test dataset
test_mAP = evaluate_model_infected(test_set_infected, model_infected, infected
Cfg)
print("Test mAP: %.3f" % test_mAP)

print("\n\nTime taken to run this cell :", datetime.now() - start)
```

```
Train mAP: 0.910
Test mAP: 0.911


Time taken to run this cell : 0:11:34.431486
```

Combined mAP for Both Models :

In [0]:
```
train_mAP = 0.832
test_mAP = 0.853
train_Infected_mAP = 0.910
test_Infected_mAP = 0.911

print('Combined Train mAP :',mean([train_mAP,train_Infected_mAP]))
print('Combined Test mAP :',mean([test_mAP,test_Infected_mAP]))
```

```
Combined Train mAP : 0.871
Combined Test mAP : 0.882
```

mAP for Final Model :

In [0]:
```
start = datetime.now()

# evaluate model on test dataset
test_mAP = evaluate_both_models()
print("Test mAP: %.3f" % test_mAP)

print("\n\nTime taken to run this cell :", datetime.now() - start)
```

```
Test mAP: 0.854


Time taken to run this cell : 0:16:29.941874
```

Calculating The Confusion Matrix :

In [0]:
```
def plot_confusion_matrix(cm,
                          target_names,
                          title='Confusion matrix',
                          cmap=None,
                          normalize=True):
    """
    given a sklearn confusion matrix (cm), make a nice plot

    Arguments
    ---------
    cm:           confusion matrix from sklearn.metrics.confusion_matrix

    target_names: given classification classes such as [0, 1, 2]
                  the class names, for example: ['high', 'medium', 'low']

    title:        the text to display at the top of the matrix

    cmap:         the gradient of the values displayed from matplotlib.pyplot.
cm
                  see http://matplotlib.org/examples/color/colormaps_referenc
e.html
                  plt.get_cmap('jet') or plt.cm.Blues
```

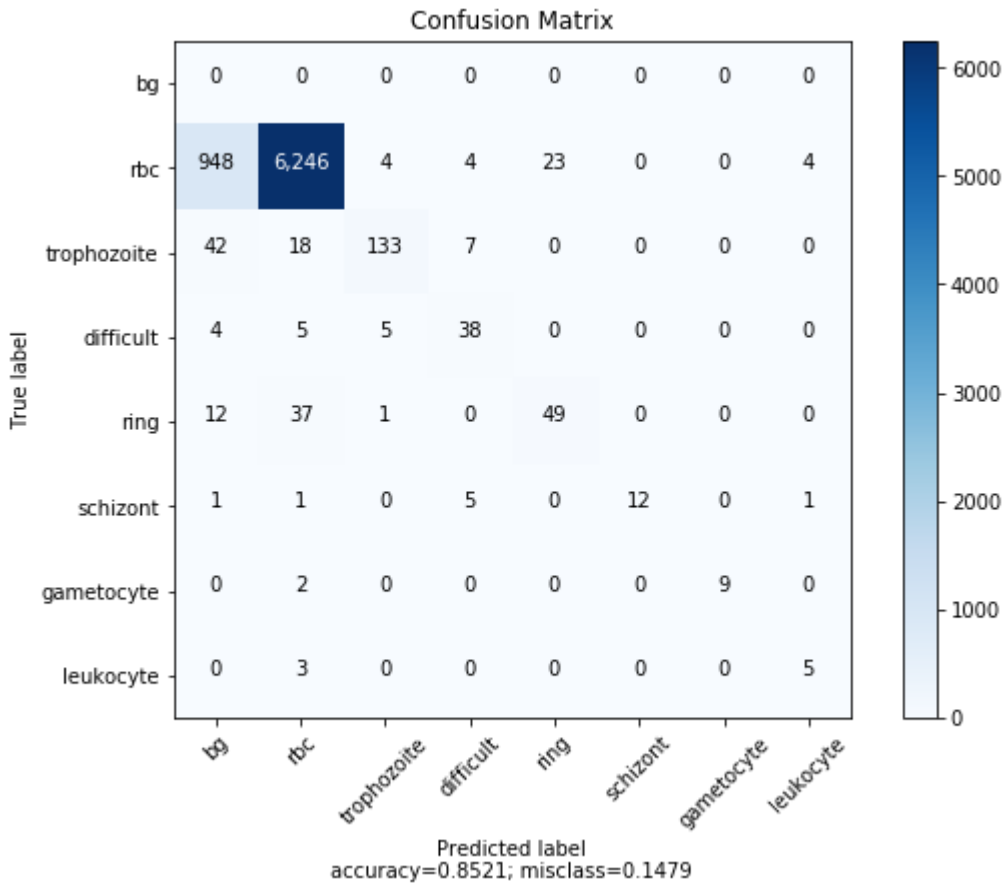
```
    normalize:    If False, plot the raw numbers
                  If True, plot the proportions

    Usage
    -----
    plot_confusion_matrix(cm           = cm,                  # confusion matr
ix created by
                                                              # sklearn.metric
s.confusion_matrix
                          normalize    = True,                # show proportio
ns
                          target_names = y_labels_vals,       # list of names
 of the classes
                          title        = best_estimator_name) # title of graph

    Citiation
    ---------
    http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusio
n_matrix.html

    """
    import matplotlib.pyplot as plt
    import numpy as np
    import itertools

    accuracy = np.trace(cm) / float(np.sum(cm))
    misclass = 1 - accuracy

    if cmap is None:
        cmap = plt.get_cmap('Blues')

    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

    if target_names is not None:
        tick_marks = np.arange(len(target_names))
        plt.xticks(tick_marks, target_names, rotation=45)
        plt.yticks(tick_marks, target_names)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]


    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, "{:,}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
```

```
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label\naccuracy={:0.4f}; misclass={:0.4f}'.format(ac
curacy, misclass))
    plt.show()

def calc_iou(gt_bbox, pred_bbox):
    '''
    This function takes the predicted bounding box and ground truth bounding b
ox and
    return the IoU ratio
    '''
    x_topleft_gt, y_topleft_gt, x_bottomright_gt, y_bottomright_gt= gt_bbox
    x_topleft_p, y_topleft_p, x_bottomright_p, y_bottomright_p= pred_bbox

    if (x_topleft_gt > x_bottomright_gt) or (y_topleft_gt> y_bottomright_gt):
        raise AssertionError("Ground Truth Bounding Box is not correct")
    if (x_topleft_p > x_bottomright_p) or (y_topleft_p> y_bottomright_p):
        raise AssertionError("Predicted Bounding Box is not correct",x_topleft
_p, x_bottomright_p,y_topleft_p,y_bottomright_gt)


    #if the GT bbox and predcited BBox do not overlap then iou=0
    if(x_bottomright_gt< x_topleft_p):
        # If bottom right of x-coordinate  GT  bbox is less than or above the
 top left of x coordinate of  the predicted BBox

        return 0.0
    if(y_bottomright_gt< y_topleft_p):  # If bottom right of y-coordinate  GT
bbox is less than or above the top left of y coordinate of  the predicted BBox

        return 0.0
    if(x_topleft_gt> x_bottomright_p): # If bottom right of x-coordinate  GT
 bbox is greater than or below the bottom right  of x coordinate of  the predc
ited BBox

        return 0.0
    if(y_topleft_gt> y_bottomright_p): # If bottom right of y-coordinate  GT
 bbox is greater than or below the bottom right  of y coordinate of  the predc
ited BBox

        return 0.0


    GT_bbox_area = (x_bottomright_gt -  x_topleft_gt + 1) * (  y_bottomright_g
t -y_topleft_gt + 1)
    Pred_bbox_area =(x_bottomright_p - x_topleft_p + 1 ) * ( y_bottomright_p -
y_topleft_p + 1)

    x_top_left =np.max([x_topleft_gt, x_topleft_p])
    y_top_left = np.max([y_topleft_gt, y_topleft_p])
    x_bottom_right = np.min([x_bottomright_gt, x_bottomright_p])
    y_bottom_right = np.min([y_bottomright_gt, y_bottomright_p])

    intersection_area = (x_bottom_right- x_top_left + 1) * (y_bottom_right-y_t
op_left  + 1)
```

```
    union_area = (GT_bbox_area + Pred_bbox_area - intersection_area)

    return intersection_area/union_area
```

In [0]:
```
start = datetime.now()

#creating confusion matrix
confusion_mat = np.zeros((len(test_set.class_ids),len(test_set.class_ids)),dty
pe=int)

for image_id in tqdm(test_set.image_ids):

  image, _, gt_class_id, gt_bbox, gt_mask = load_image_gt(test_set, cfg, image
_id, use_mini_mask=False)
  scaled_image = mold_image(image, cfg)
  sample = expand_dims(scaled_image, 0)
  yhat = model.detect(sample, verbose=0)
  r = yhat[0]

  original_image, _, gt_class_id_inf, gt_bbox_inf, gt_mask_inf =load_image_gt(
test_set_infected, infectedCfg, image_id, use_mini_mask=False)
  results = model_infected.detect([original_image], verbose=0)
  r2 = results[0]

  roisr1r2 = np.concatenate((r['rois'],r2['rois']))
  try:
    masksr1r2 = tf.concat((r['masks'],r2['masks']),axis=2).eval(session=tf.com
pat.v1.Session())
  except:
    masksr1r2 = r['masks']
  classIdsr1r2 = np.concatenate((r['class_ids'],r2['class_ids']))
  scoresr1r2 = np.concatenate((r['scores'],r2['scores']))

  for igb, gt_box in enumerate(gt_bbox):
    box_found = False
    for ipb, pred_box in enumerate(roisr1r2):
      iou= calc_iou(gt_box, pred_box)
      if iou > 0.65:
        box_found=True
        actual_class = gt_class_id[igb]
        pred_class = classIdsr1r2[ipb]
        confusion_mat[actual_class][pred_class] += 1
    if(not box_found):
      confusion_mat[actual_class][0] += 1

plot_confusion_matrix(cm=confusion_mat, normalize=False,
                      target_names=['bg', 'rbc', 'trophozoite', 'difficult',
'ring', 'schizont', 'gametocyte', 'leukocyte'],
                      title="Confusion Matrix")

print("\n\nTime taken to run this cell :", datetime.now() - start)
```

100%|██████████| 120/120 [19:05<00:00, 10.34s/it]



Time taken to run this cell : 0:19:05.755781

F1 Score For Each Class :

In [0]:
```
#True positive: diagonal position, cm(x, x).
#False positive: sum of column x (without main diagonal), sum(cm(:, x))-cm(x,
 x).
#False negative: sum of row x (without main diagonal), sum(cm(x, :), 2)-cm(x,
 x).
#precision=TP / (TP + FP)
#recall = TP / (TP + FN)
#f1 = 2 * (precision*recall) / (precision+recall)

def calculate_precision_recall_f1(cell_name,confusion_mat):
  class_mapping = {'rbc': 1, 'trophozoite':2, 'difficult':3, 'ring':4, 'schizo
nt':5, 'gametocyte':6, 'leukocyte':7}

  idx = class_mapping[cell_name]

  tp = confusion_mat[idx,idx]
  fp = sum(confusion_mat[:, idx]) - tp
  fn = sum(confusion_mat[idx, :]) - tp

  precision = tp/(tp+fp)
  recall = tp/(tp+fn)
  f1 = 2 * ((precision*recall)/(precision+recall))

  return round(precision,2), round(recall,2), round(f1,2)
```

In [0]:
```
x = PrettyTable()

x.field_names = ["Cell Name", "Precision", "Recall", "F1-Score"]

precision, recall, f1 = calculate_precision_recall_f1('rbc',confusion_mat)
x.add_row(["rbc", precision, recall, f1])

precision, recall, f1 = calculate_precision_recall_f1('trophozoite',confusion_
mat)
x.add_row(["trophozoite", precision, recall, f1])

precision, recall, f1 = calculate_precision_recall_f1('difficult',confusion_ma
t)
x.add_row(["difficult", precision, recall, f1])

precision, recall, f1 = calculate_precision_recall_f1('ring',confusion_mat)
x.add_row(["ring", precision, recall, f1])

precision, recall, f1 = calculate_precision_recall_f1('schizont',confusion_mat
)
x.add_row(["schizont", precision, recall, f1])

precision, recall, f1 = calculate_precision_recall_f1('gametocyte',confusion_m
at)
x.add_row(["gametocyte", precision, recall, f1])

precision, recall, f1 = calculate_precision_recall_f1('leukocyte',confusion_ma
t)
x.add_row(["leukocyte", precision, recall, f1])

print(x)
```

```
+-------------+-----------+--------+----------+
|  Cell Name  | Precision | Recall | F1-Score |
+-------------+-----------+--------+----------+
|     rbc     |    0.99   |  0.86  |   0.92   |
| trophozoite |    0.93   |  0.66  |   0.78   |
|  difficult  |    0.7    |  0.73  |   0.72   |
|     ring    |    0.68   |  0.49  |   0.57   |
|   schizont  |    1.0    |  0.6   |   0.75   |
|  gametocyte |    1.0    |  0.82  |   0.9    |
|  leukocyte  |    0.5    |  0.62  |   0.56   |
+-------------+-----------+--------+----------+
```

Detect Malaria Cells in New Photos

Visualize : Prediction on Train & Test Data

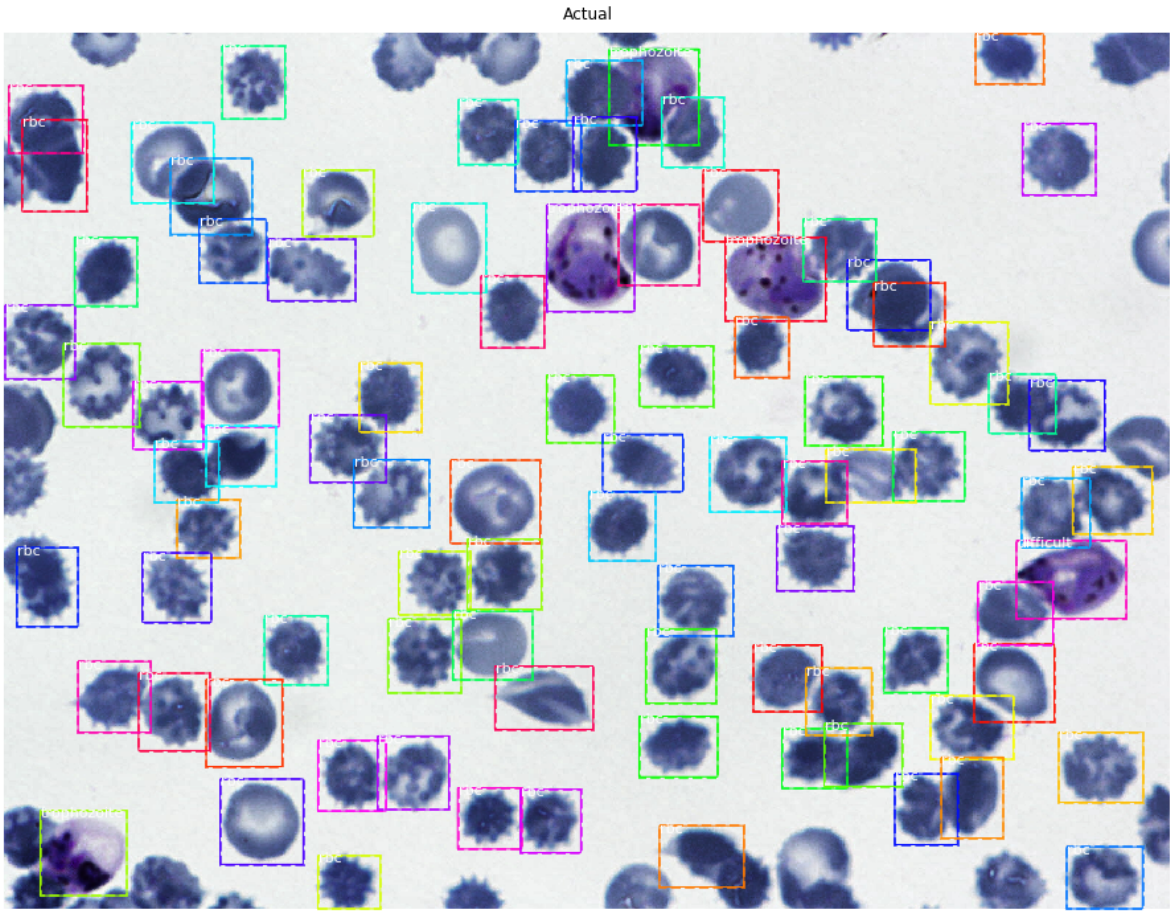
In [0]:
```
def show_Actual(image_id):
  dataset = train_set

  image = dataset.load_image(image_id)
  mask, class_ids = dataset.load_mask(image_id)
  bbox = extract_bboxes(mask)
  display_instances(image, bbox, mask, class_ids, dataset.class_names, show_ma
sk=False, title='Actual')

def detect_all(image_id):
  dataset = train_set
  original_image, image_meta, gt_class_id, gt_bbox, gt_mask =load_image_gt(dat
aset, cfg, image_id, use_mini_mask=False)
  results = model.detect([original_image], verbose=1)
  results2 = model_infected.detect([original_image], verbose=1)
  r = results[0]
  r2 = results2[0]
  roisr1r2 = np.concatenate((r['rois'],r2['rois']))
  masksr1r2 = tf.concat((r['masks'],r2['masks']),axis=2).eval(session=tf.compa
t.v1.Session())
  classIdsr1r2 = np.concatenate((r['class_ids'],r2['class_ids']))
  scoresr1r2 = np.concatenate((r['scores'],r2['scores']))
  display_instances(original_image, roisr1r2, masksr1r2, classIdsr1r2, dataset
.class_names, scoresr1r2, show_mask=False, title='Predicted All')

def detect_infected(image_id):
  dataset = train_set_infected

  original_image, image_meta, gt_class_id, gt_bbox, gt_mask =load_image_gt(dat
aset, infectedCfg, image_id, use_mini_mask=False)
  results = model_infected.detect([original_image], verbose=1)
  r = results[0]
  roisr1r2 = r['rois']
  masksr1r2 = r['masks']
  classIdsr1r2 = r['class_ids']
  scoresr1r2 = r['scores']
  display_instances(original_image, roisr1r2, masksr1r2, classIdsr1r2, dataset
.class_names, scoresr1r2, show_mask=False, title='Predicted Infected')

def detect_malaria(image_id):
  dataset = train_set

  original_image, image_meta, gt_class_id, gt_bbox, gt_mask =load_image_gt(dat
aset, cfg, image_id, use_mini_mask=False)
  results = model.detect([original_image], verbose=1)
  r = results[0]
  roisr1r2 = r['rois']
  masksr1r2 = r['masks']
  classIdsr1r2 = r['class_ids']
  scoresr1r2 = r['scores']
  display_instances(original_image, roisr1r2, masksr1r2, classIdsr1r2, dataset
.class_names, scoresr1r2, show_mask=False, title='Predicted RBC')
```

Prediction on Train Data : Existing Seen Data

Predicting 'trophozoite'

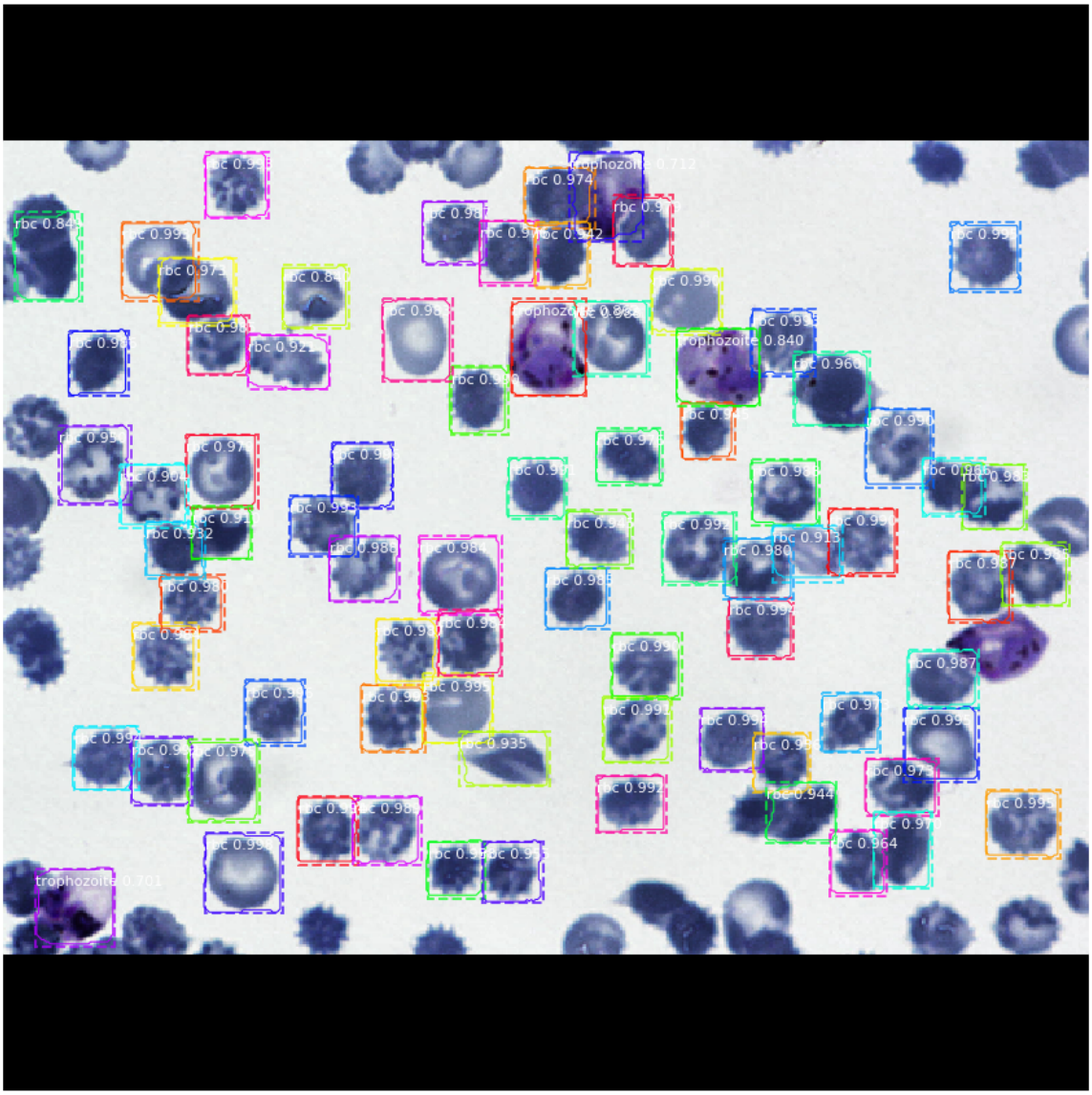
In [0]:
```
image_id = 38

print("\n Show Actual")
show_Actual(image_id)

print("\n Detecting RBC Cells")
detect_malaria(image_id)

print("\n Detecting Infected Cells")
detect_infected(image_id)

print("\n Detecting All")
detect_all(image_id)
```

Show Actual



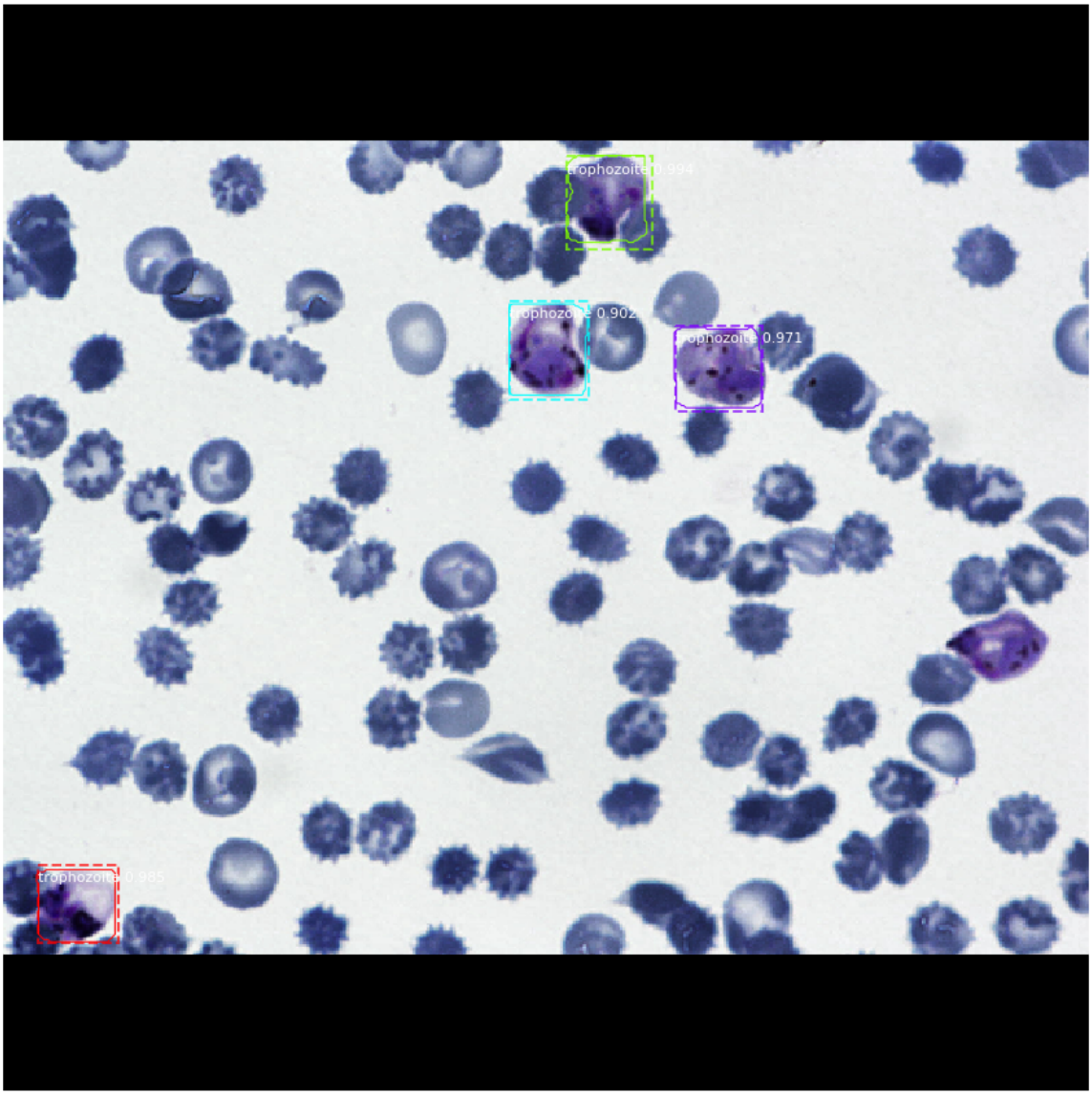

```
 Detecting RBC Cells
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
253.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
147.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
```

Predicted RBC



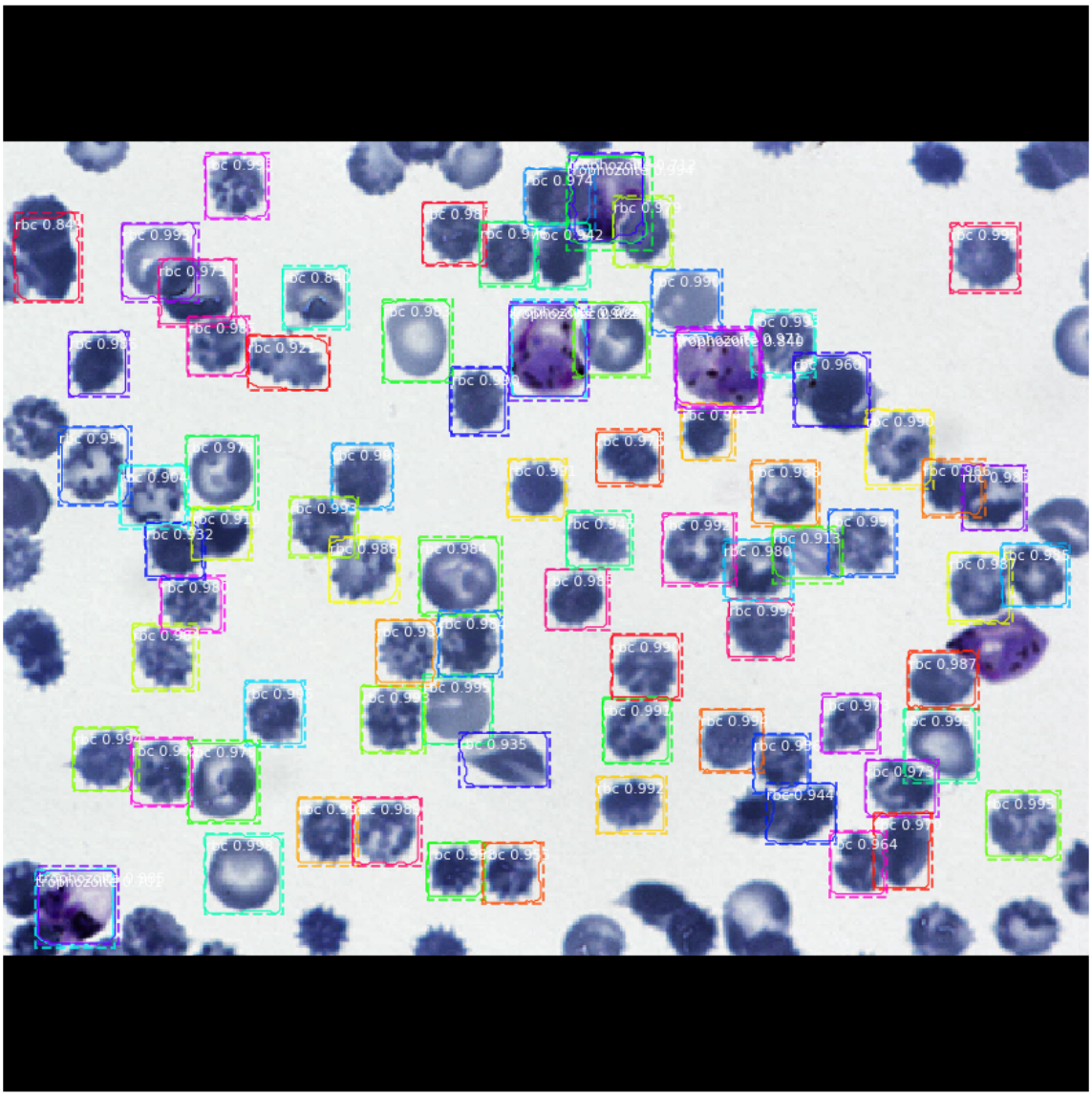
```
 Detecting Infected Cells
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
253.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
147.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
```

Predicted Infected




```
 Detecting All
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
253.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
147.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
253.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
147.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
```

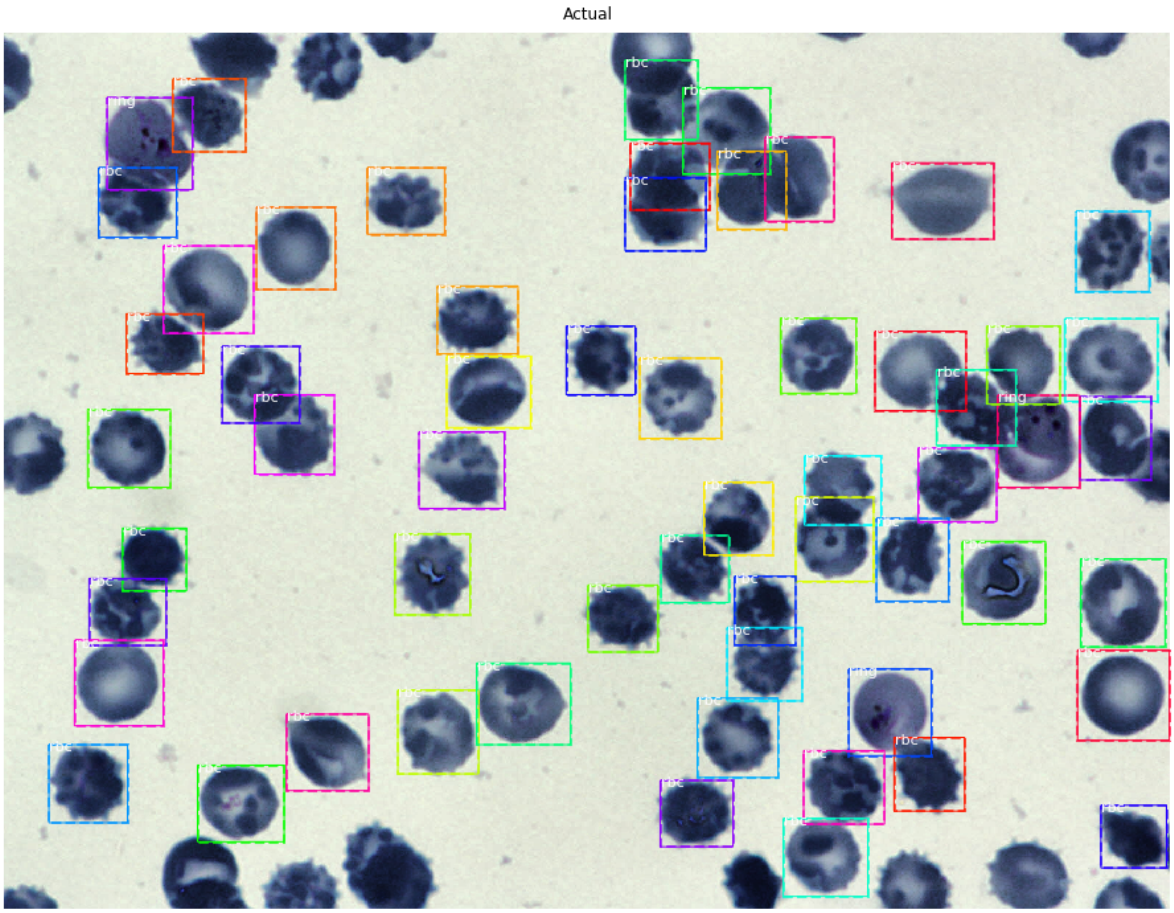
Predicted All



Predicting 'ring'

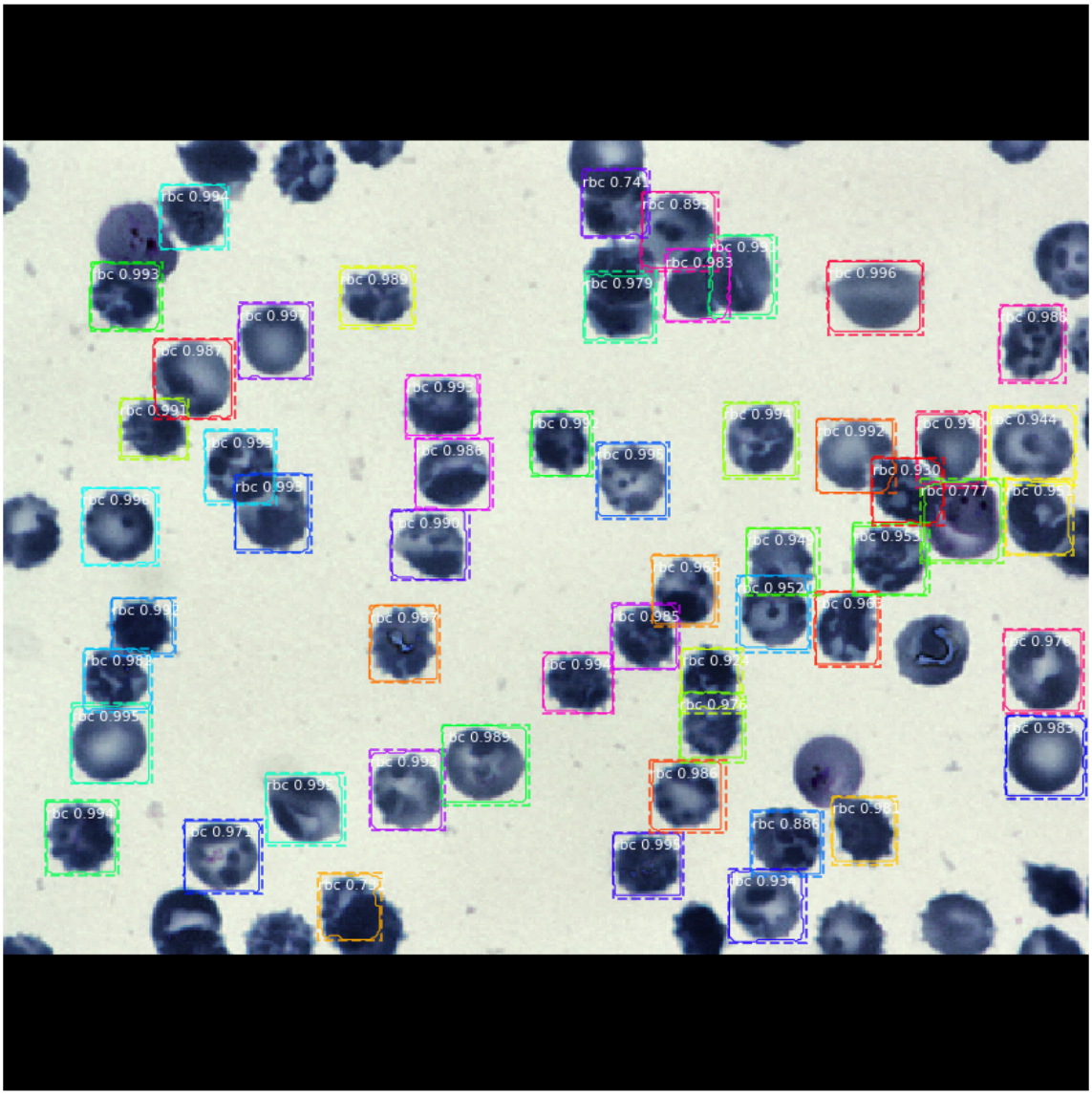
In [0]:
```
image_id = 19

print("\n Show Actual")
show_Actual(image_id)

print("\n Detecting RBC Cells")
detect_malaria(image_id)

print("\n Detecting Infected Cells")
detect_infected(image_id)

print("\n Detecting All")
detect_all(image_id)
```

Show Actual



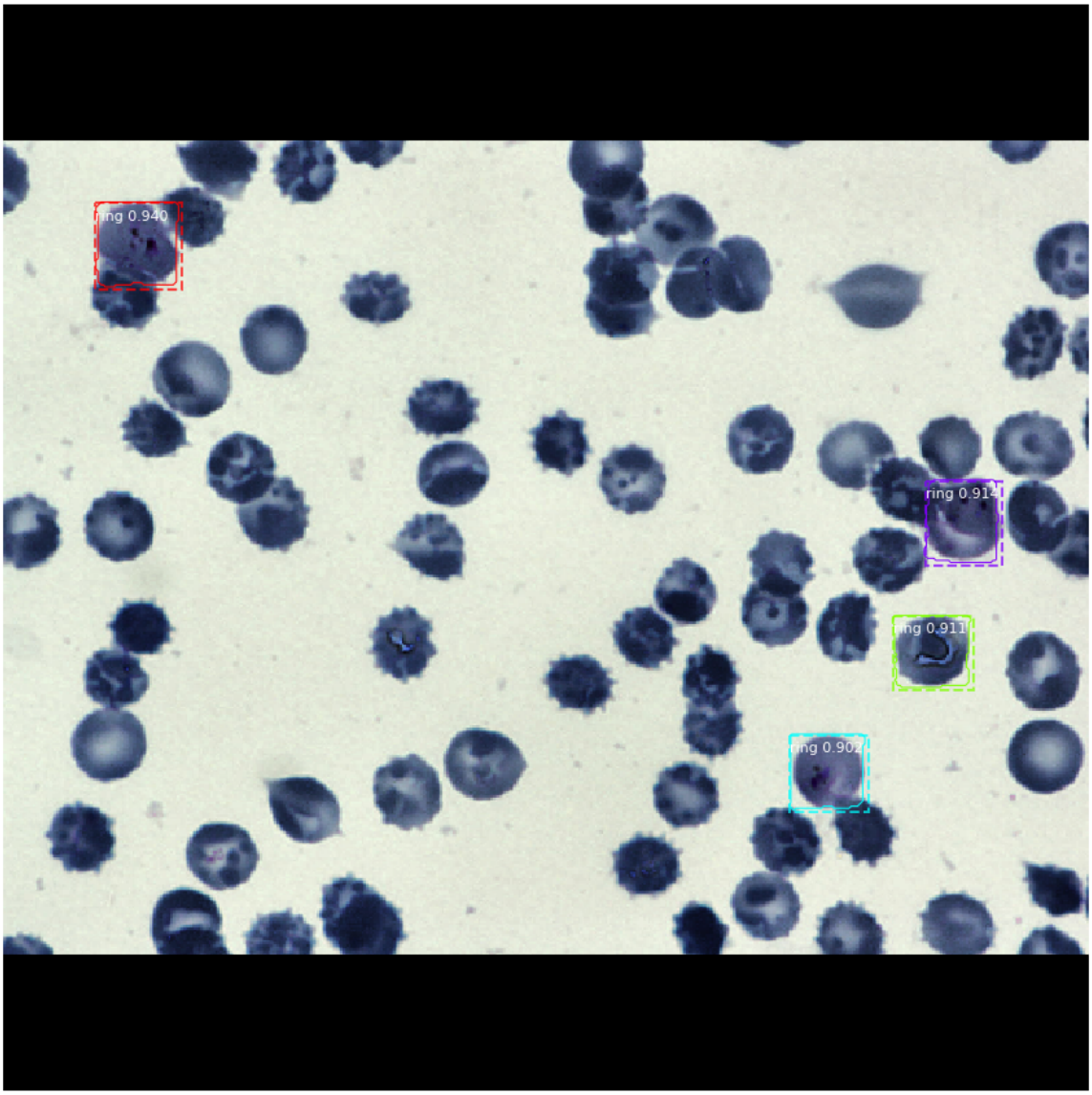

```
 Detecting RBC Cells
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
249.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
132.20000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
```

Predicted RBC

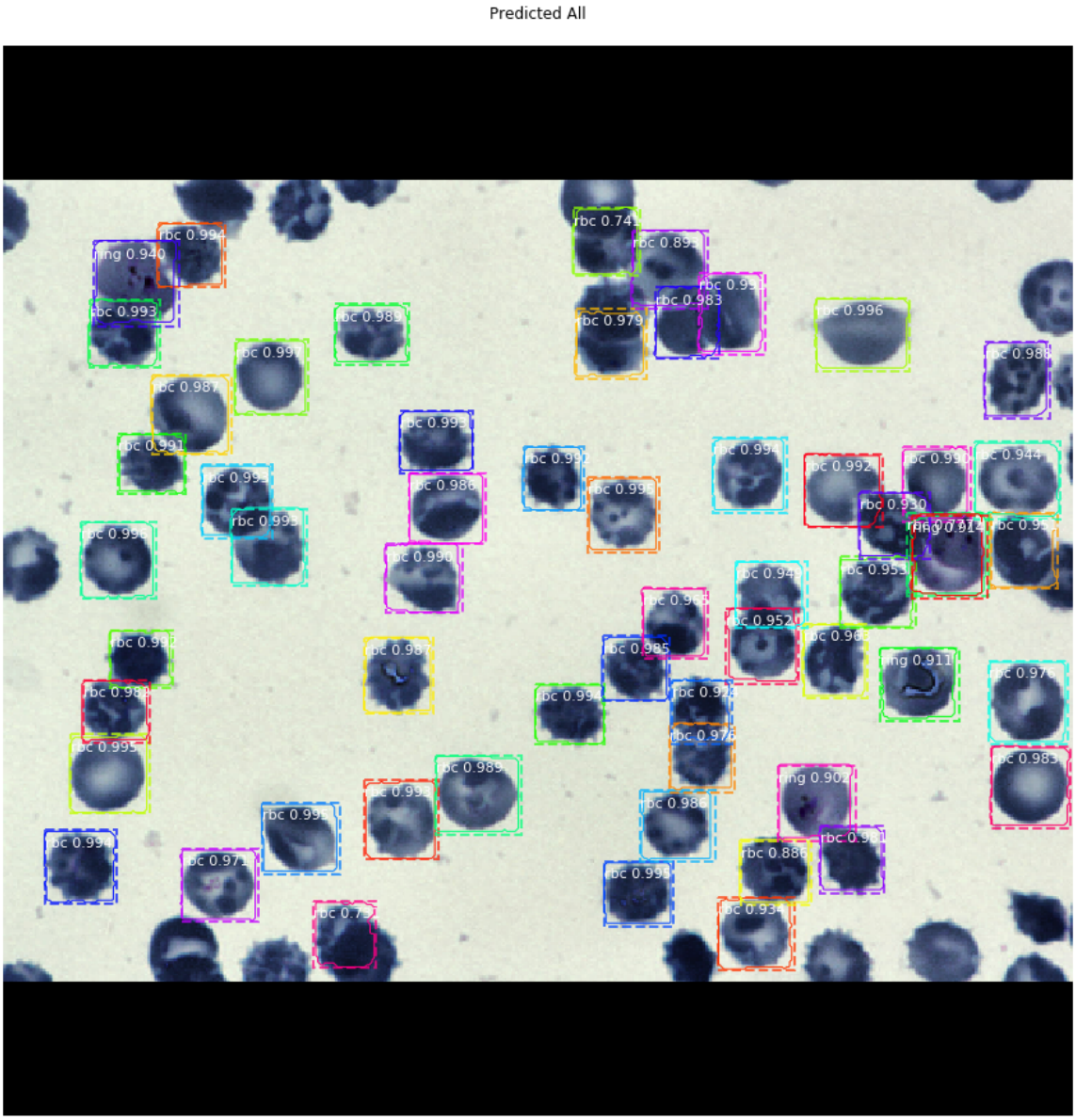


Detecting Infected Cells
Processing 1 images
image shape: (512, 512, 3) min: 0.00000 max: 249.00000 int32
molded_images shape: (1, 512, 512, 3) min: -123.70000 max: 132.20000 float64
image_metas shape: (1, 20) min: 0.00000 max: 512.00000 int64
anchors shape: (1, 65472, 4) min: -0.70849 max: 1.58325 float32

Predicted Infected



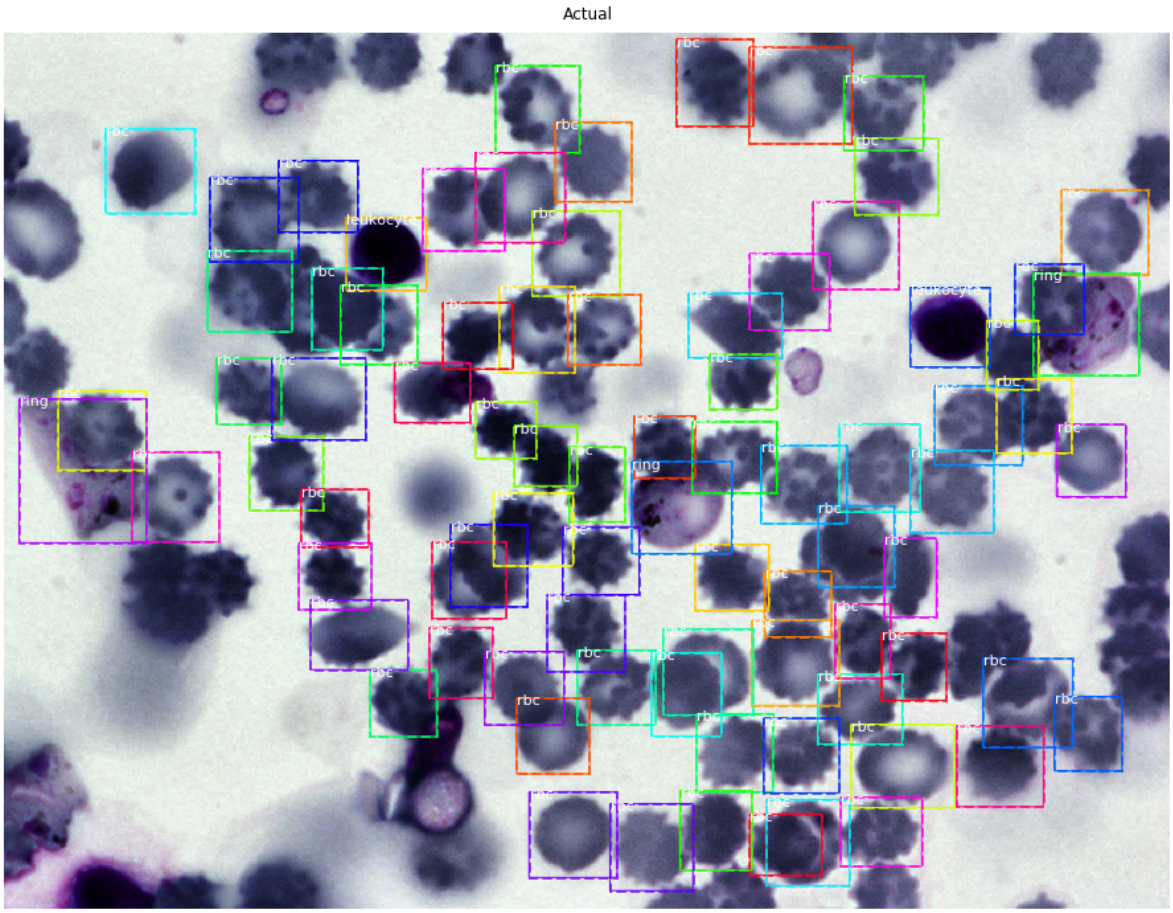

```
 Detecting All
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
249.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
132.20000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
249.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
132.20000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
```



Predicting 'leukocyte'

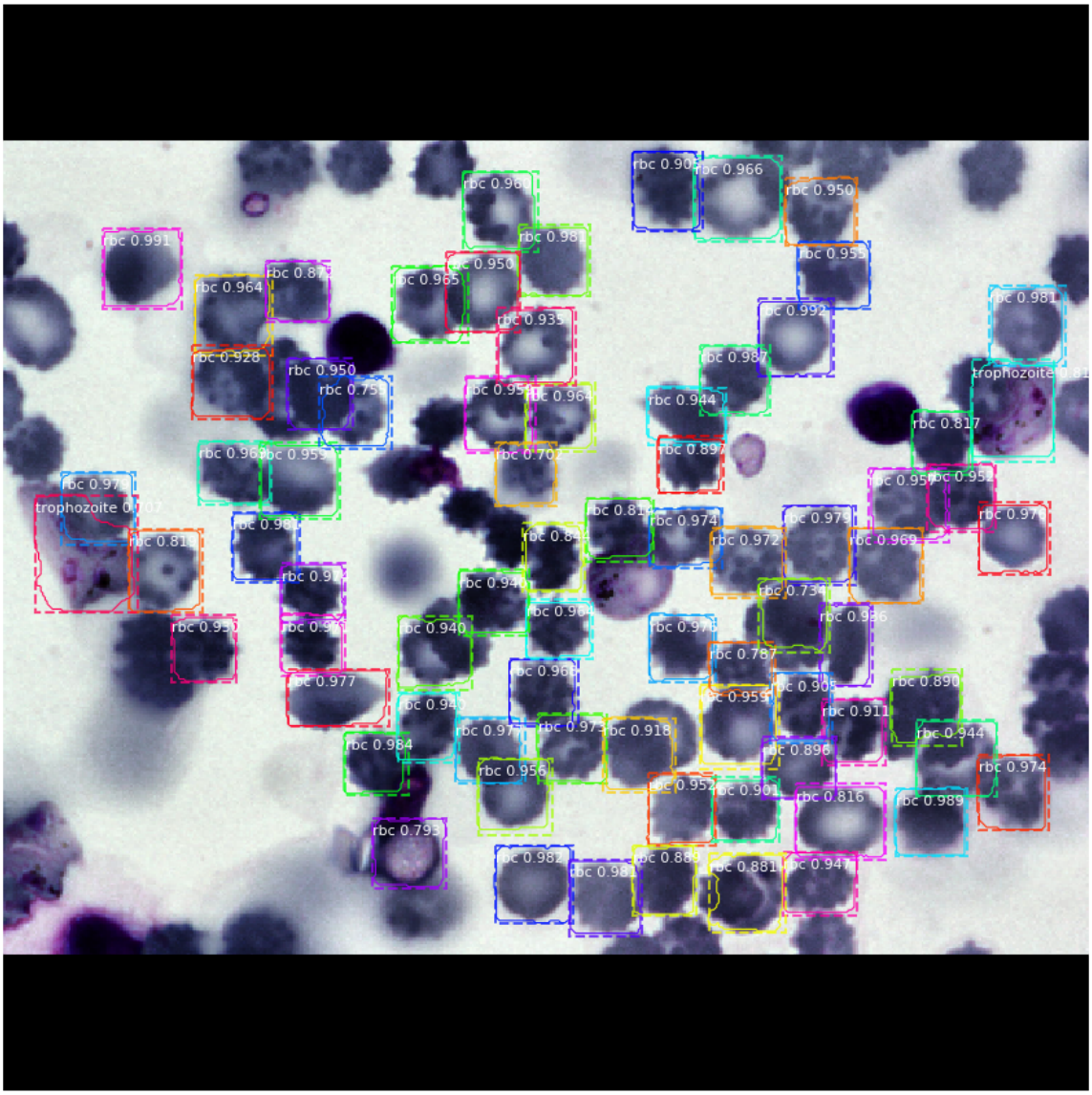
In [0]:
```
image_id = 682

print("\n Show Actual")
show_Actual(image_id)

print("\n Detecting RBC Cells")
detect_malaria(image_id)

print("\n Detecting Infected Cells")
detect_infected(image_id)

print("\n Detecting All")
detect_all(image_id)
```

Show Actual



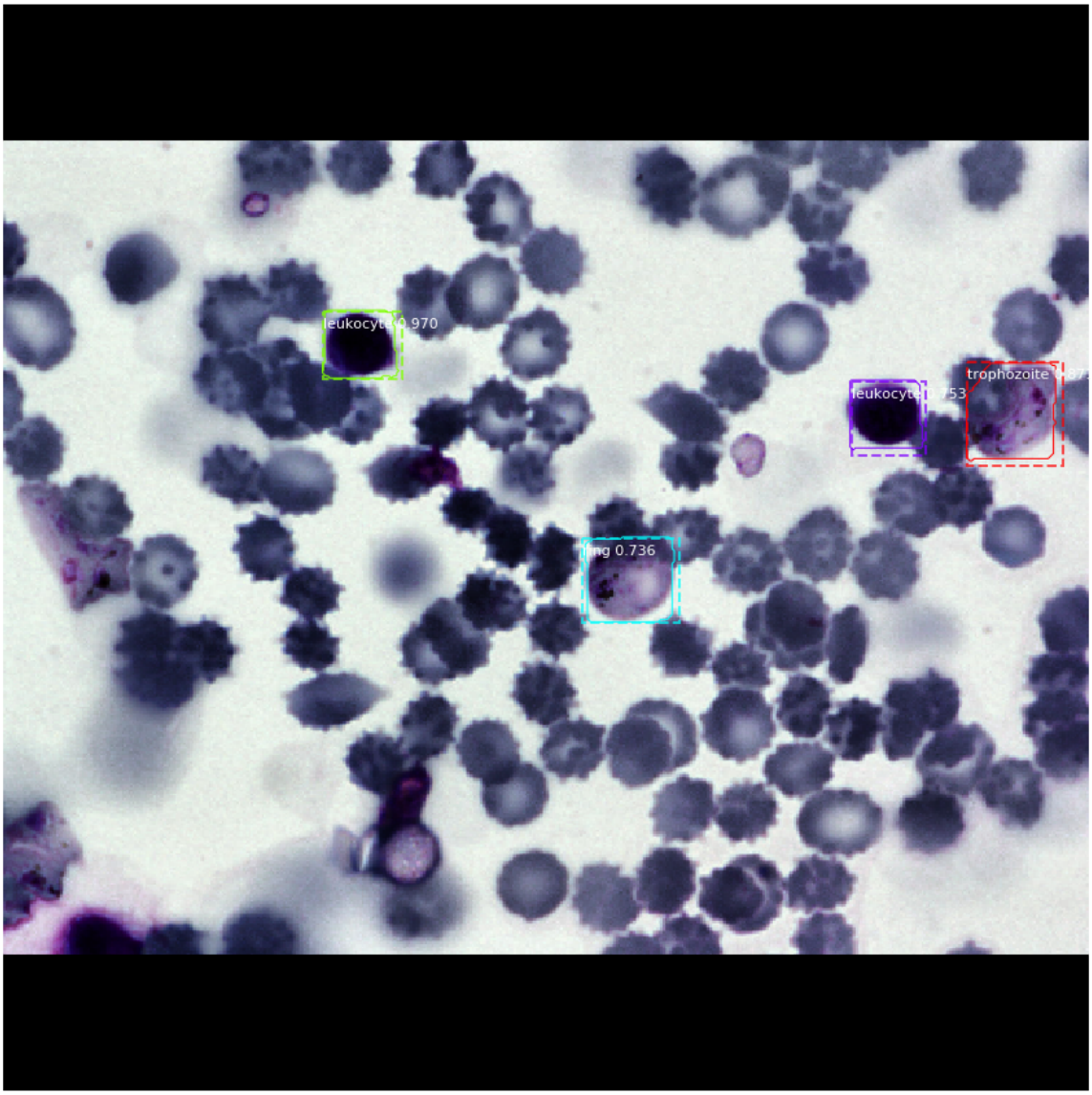

```
 Detecting RBC Cells
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
253.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
146.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
```

Predicted RBC

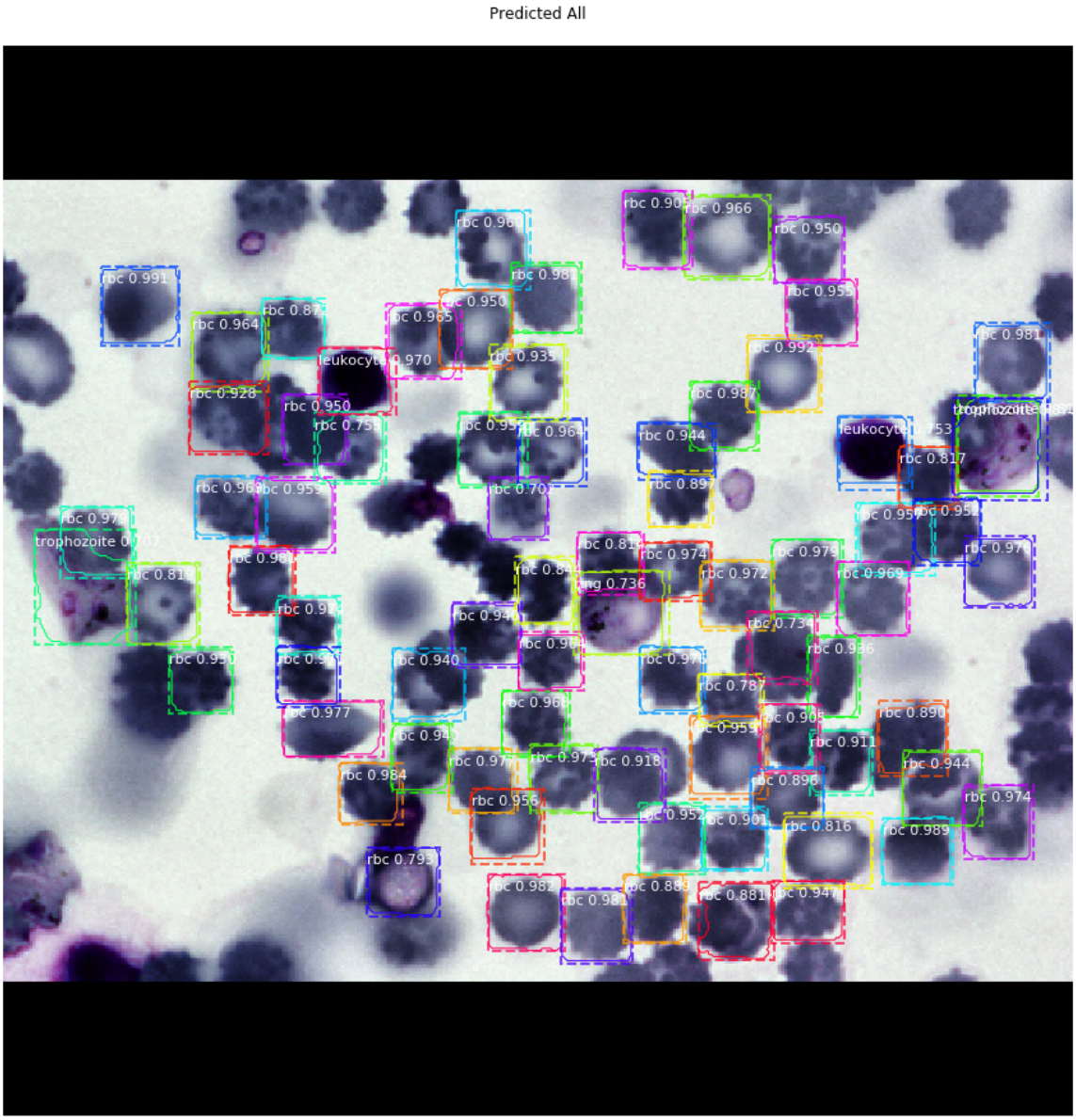



```
 Detecting Infected Cells
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
253.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
146.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
```

Predicted Infected



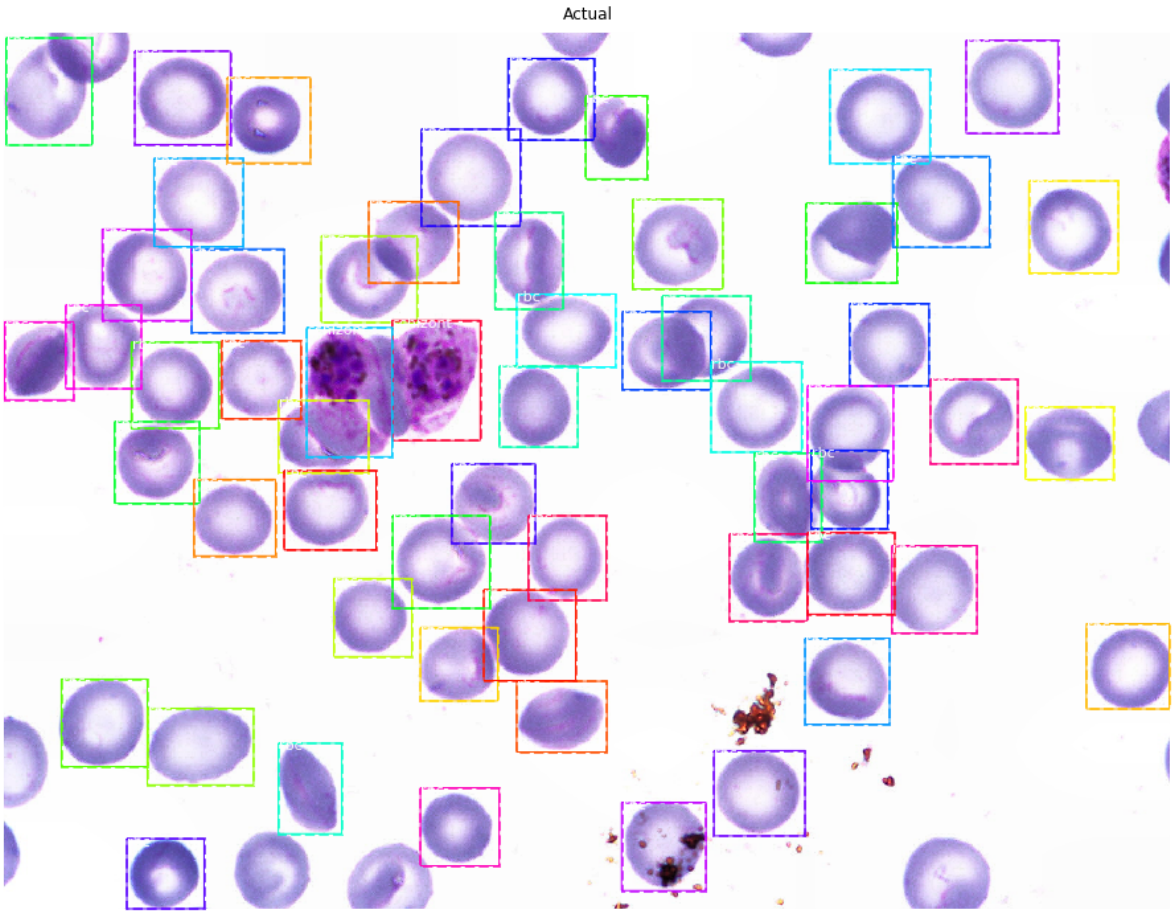

```
 Detecting All
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
253.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
146.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
253.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
146.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
```



Predicting 'schizont'

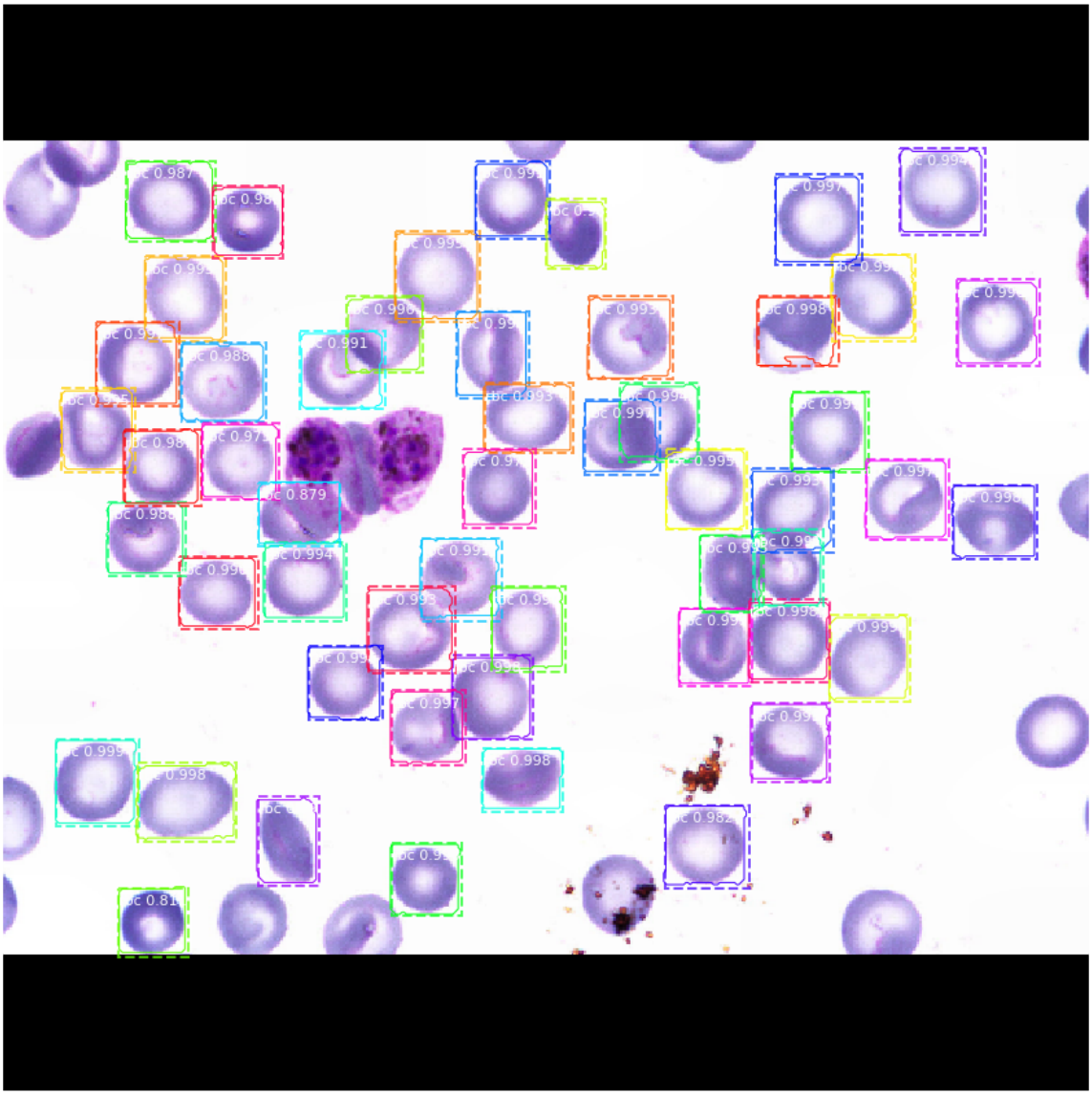
In [0]:
```
image_id = 505

print("\n Show Actual")
show_Actual(image_id)

print("\n Detecting RBC Cells")
detect_malaria(image_id)

print("\n Detecting Infected Cells")
detect_infected(image_id)

print("\n Detecting All")
detect_all(image_id)
```

Show Actual



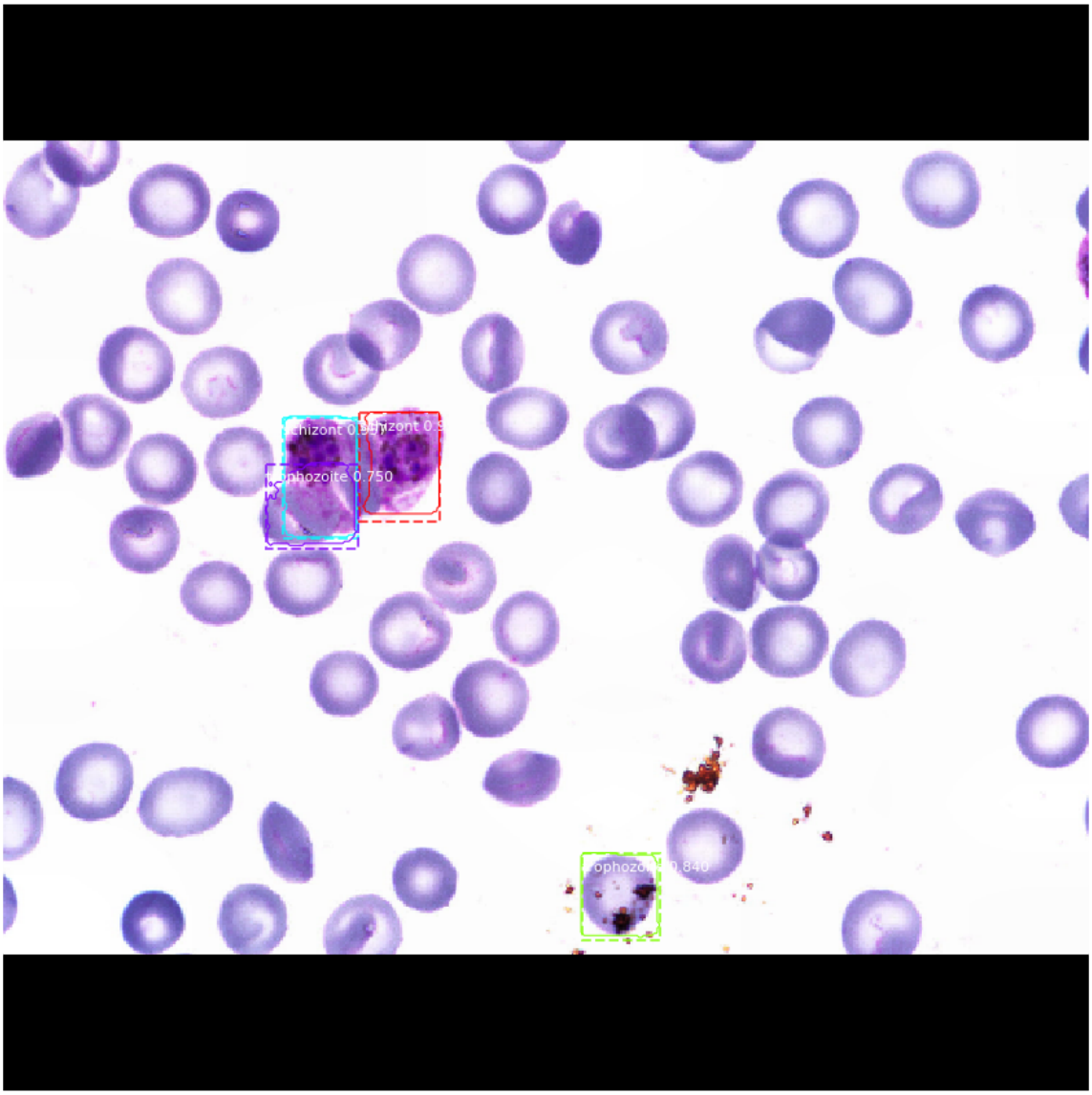

```
 Detecting RBC Cells
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
255.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
151.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
```

Predicted RBC

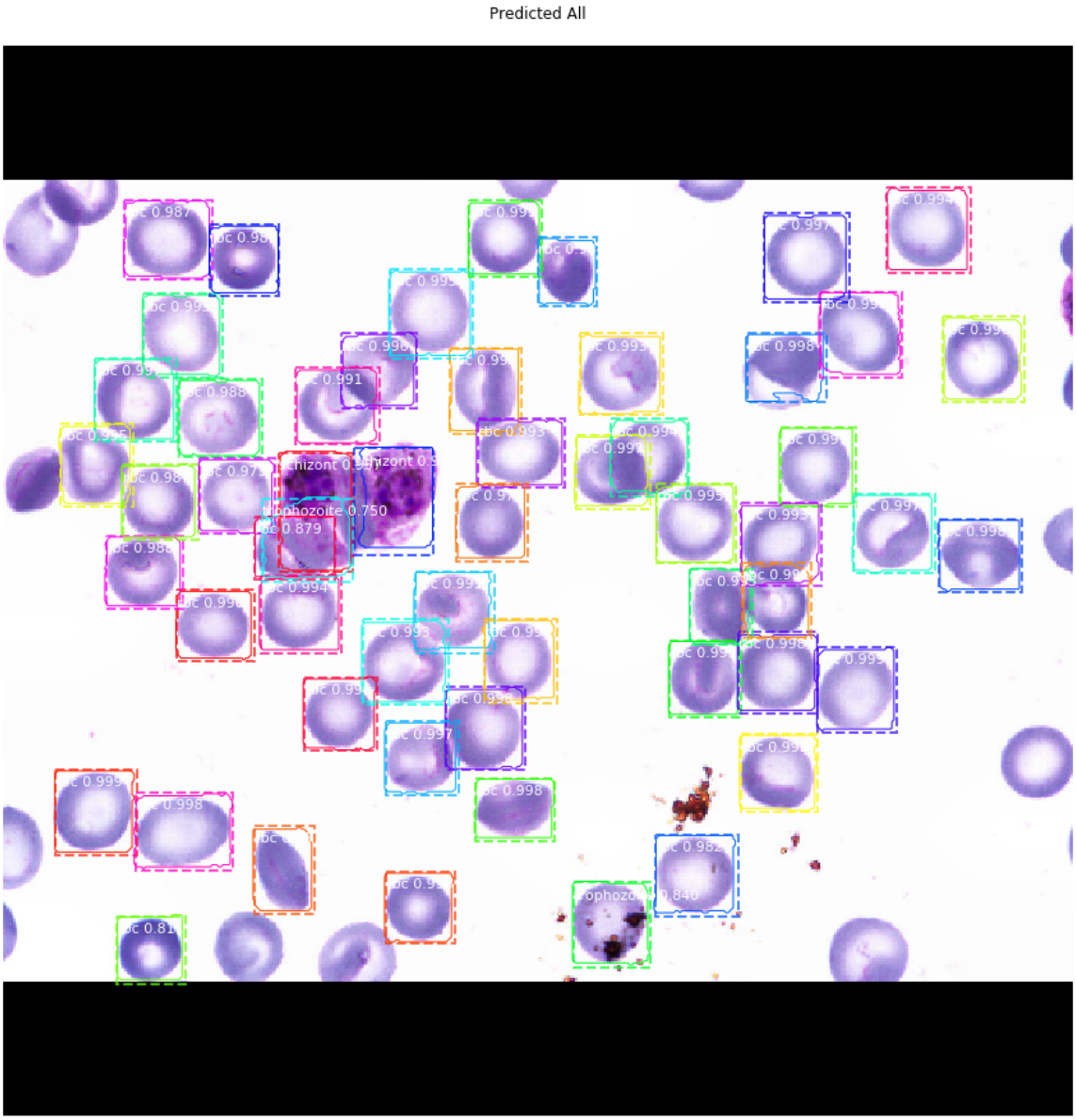


```
 Detecting Infected Cells
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
255.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
151.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
```

Predicted Infected



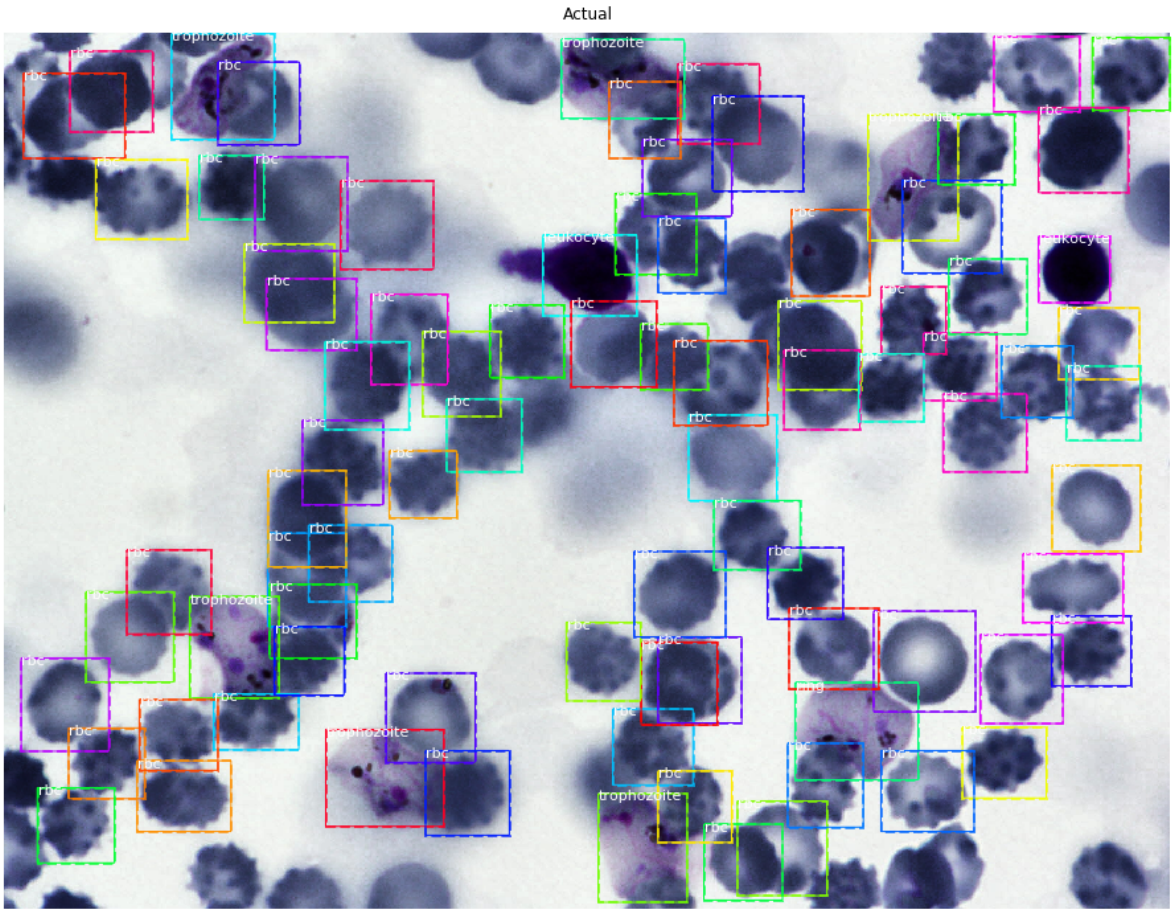

```
 Detecting All
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
255.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
151.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
255.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
151.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
```



Predicting 'multiple infections'

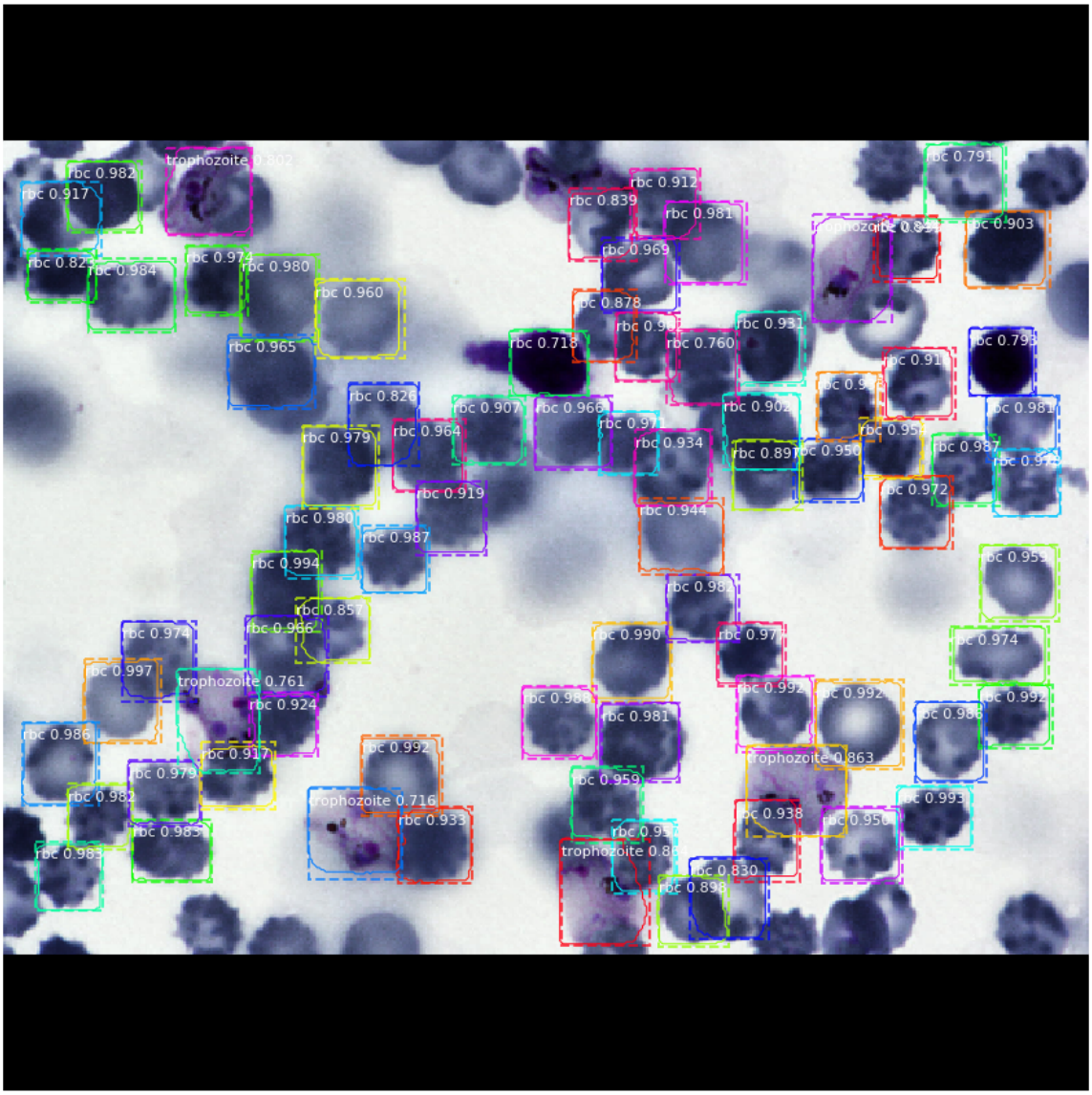
In [0]:
```
image_id = 570

print("\n Show Actual")
show_Actual(image_id)

print("\n Detecting RBC Cells")
detect_malaria(image_id)

print("\n Detecting Infected Cells")
detect_infected(image_id)

print("\n Detecting All")
detect_all(image_id)
```

Show Actual



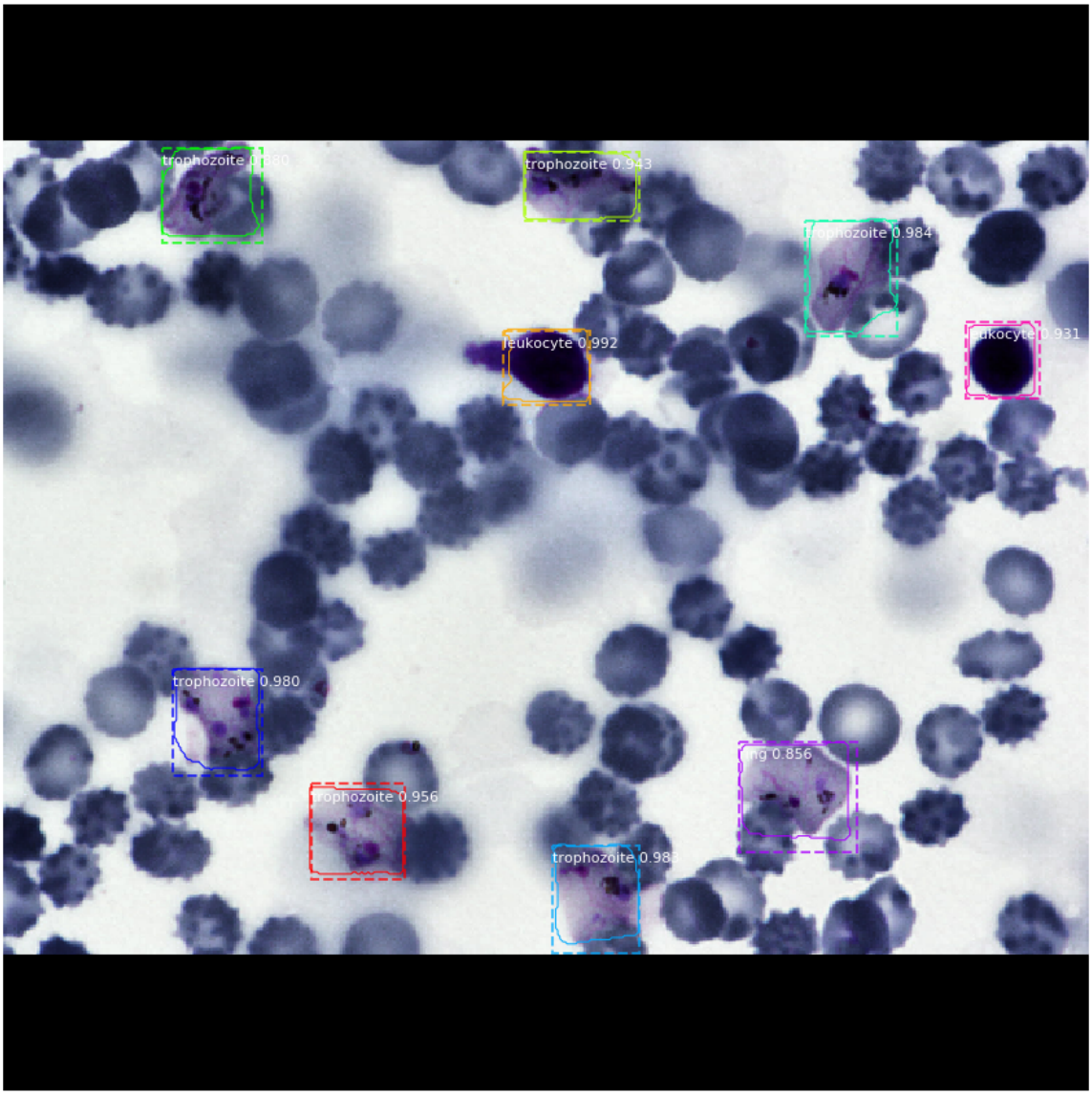

```
 Detecting RBC Cells
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
252.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
148.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
```

Predicted RBC



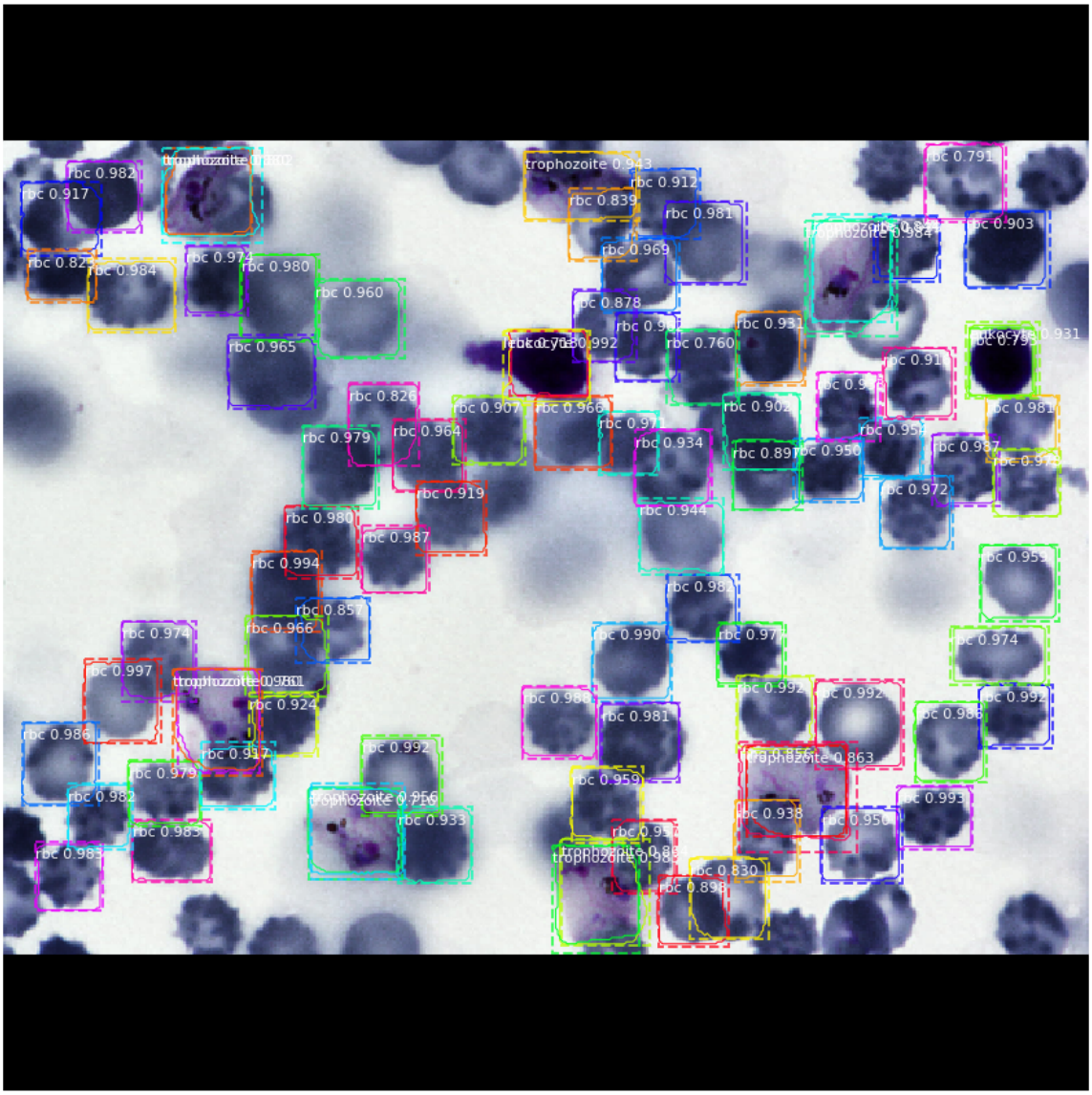

```
 Detecting Infected Cells
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
252.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
148.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
```

Predicted Infected




```
 Detecting All
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
252.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
148.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
252.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
148.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
```

Predicted All

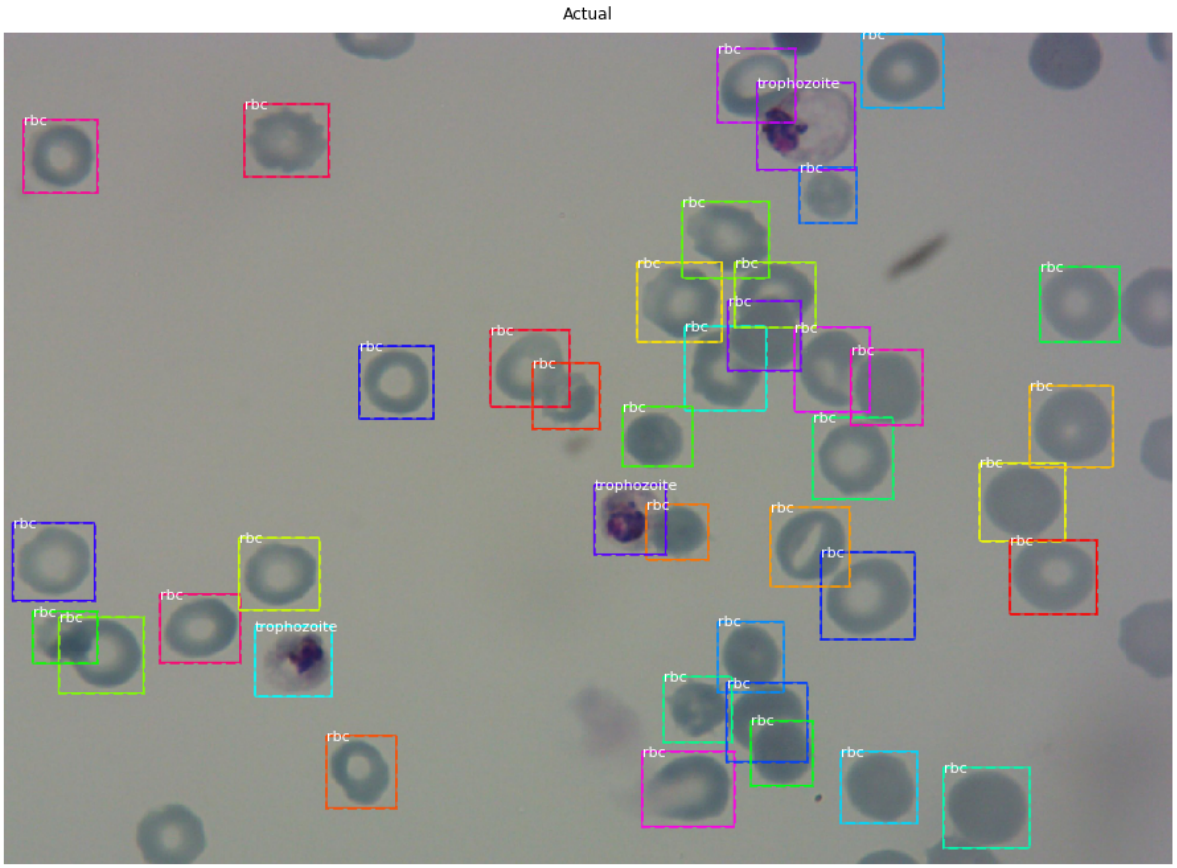


Predicting on Test Data : New Unseen Images

Predicting 'trophozoite'

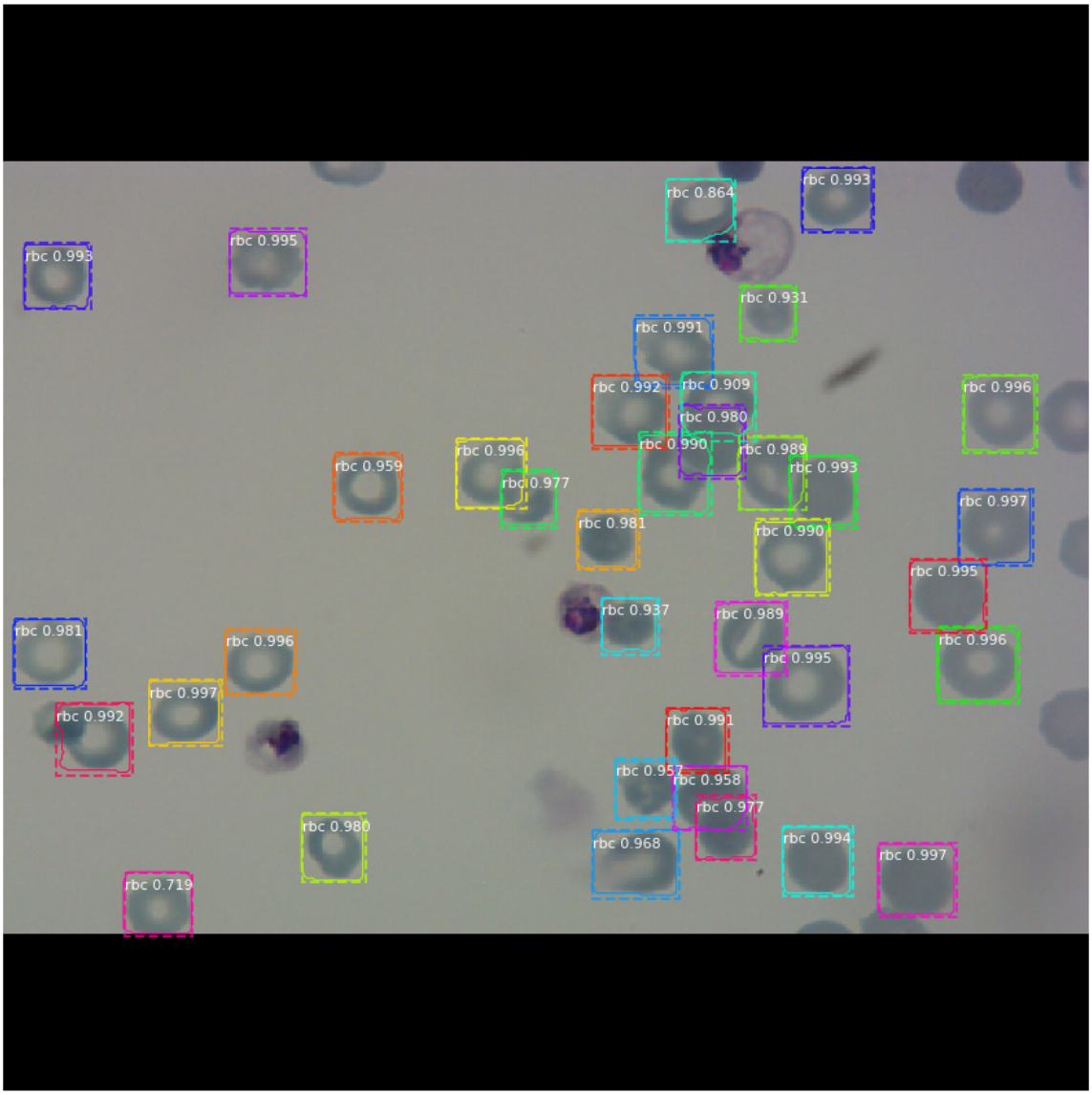
In [0]:
```
image_id = 64

print("\n Show Actual")
show_Actual(image_id)

print("\n Detecting RBC Cells")
detect_malaria(image_id)

print("\n Detecting Infected Cells")
detect_infected(image_id)

print("\n Detecting All")
detect_all(image_id)
```

Show Actual



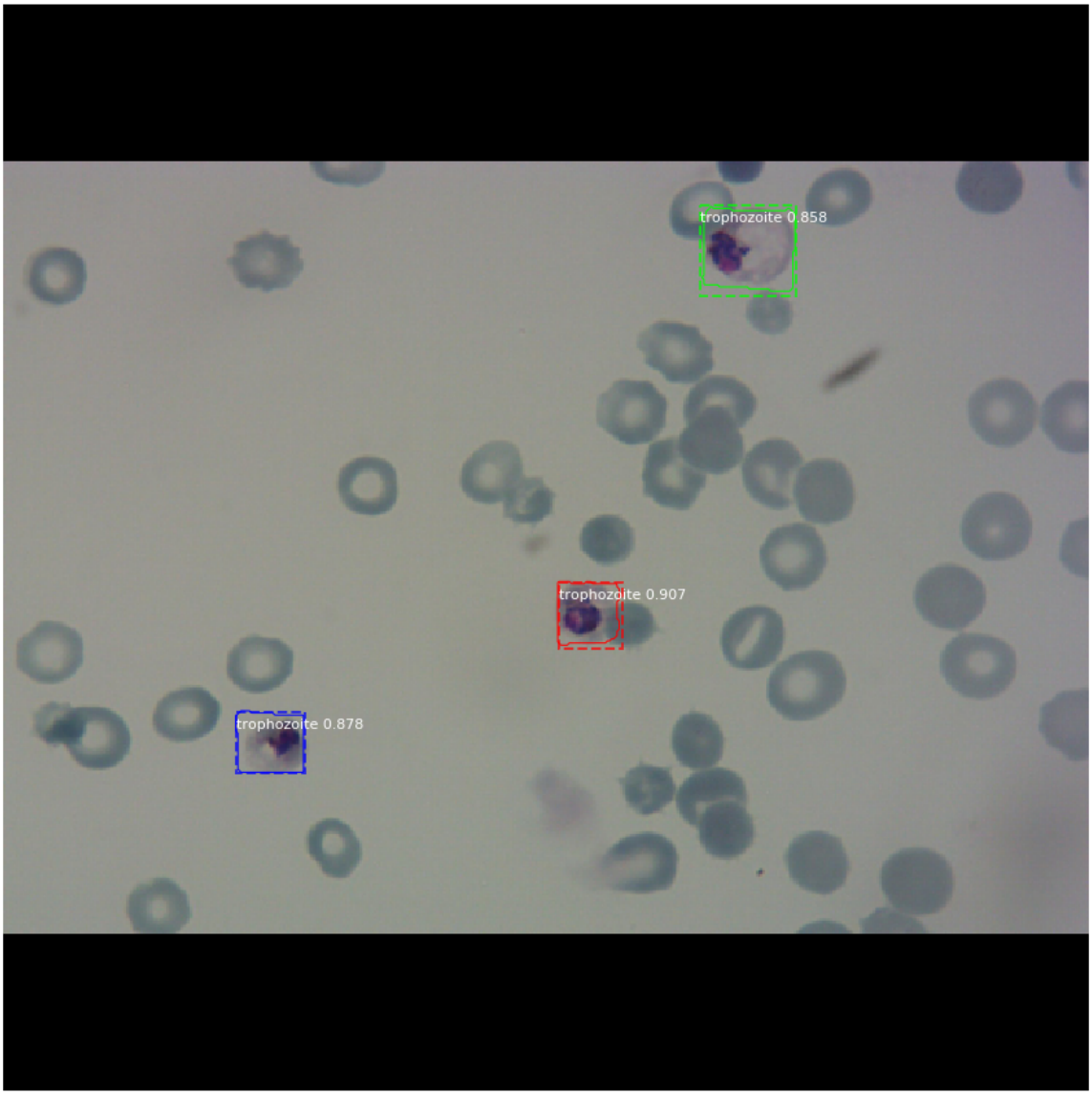

```
 Detecting RBC Cells
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
161.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
51.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
```

Predicted RBC



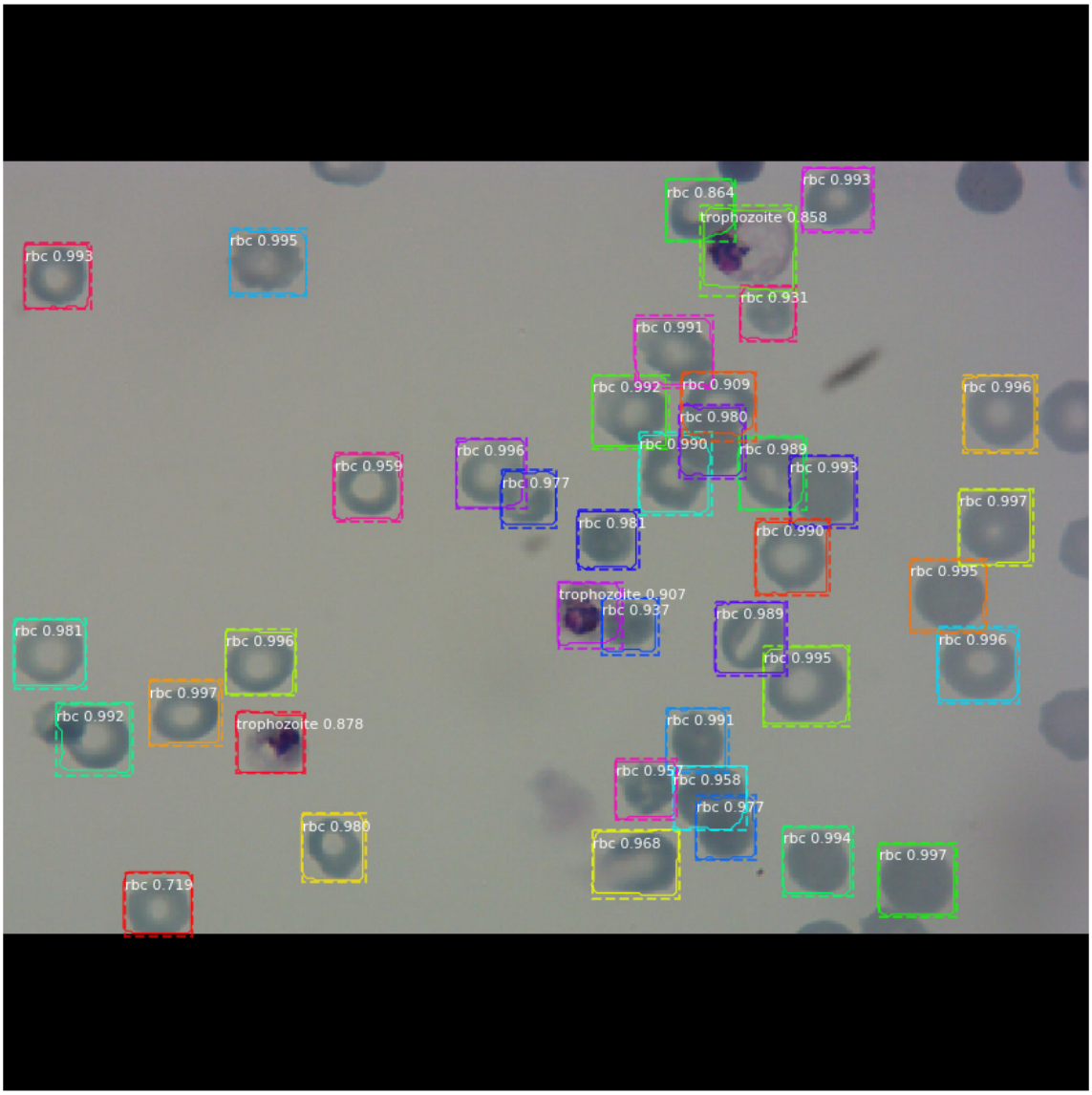

```
 Detecting Infected Cells
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
161.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
51.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
```

Predicted Infected




```
 Detecting All
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
161.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
51.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
161.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
51.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
```

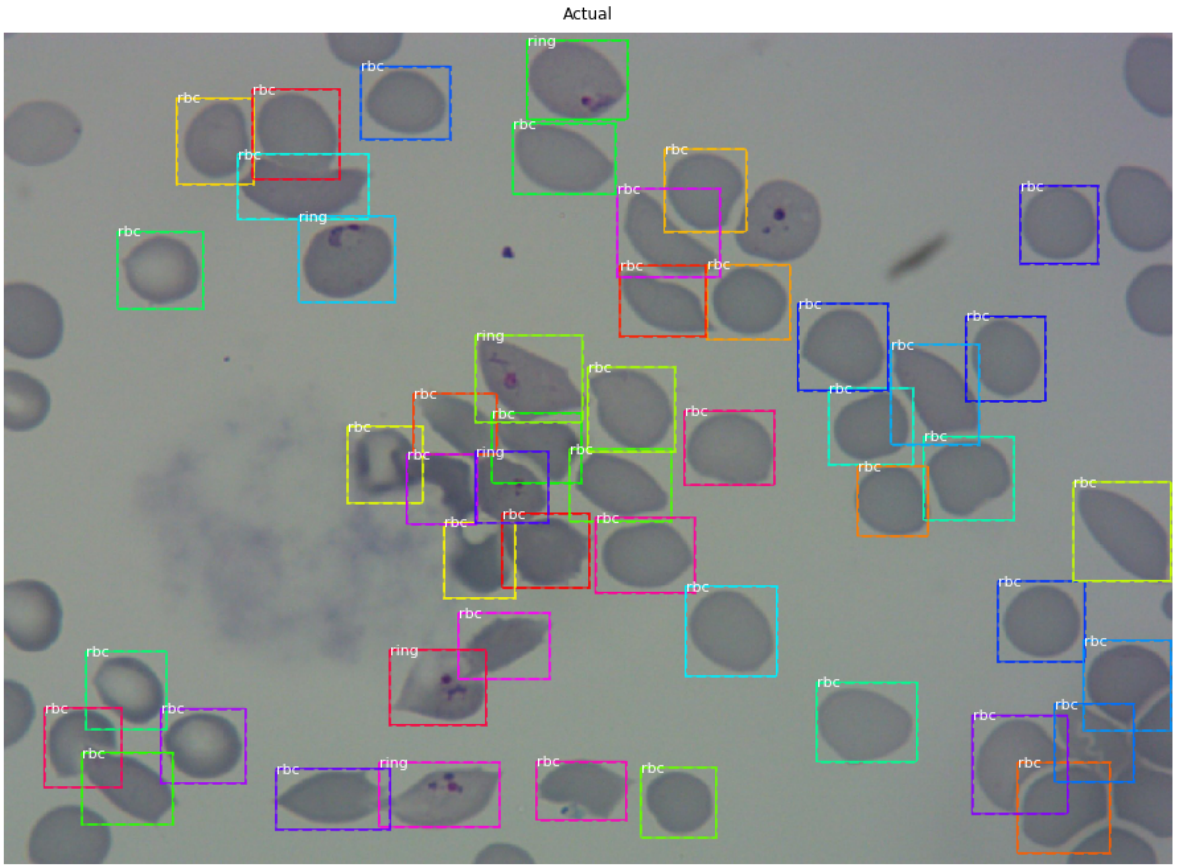
Predicted All



Predicting 'ring'

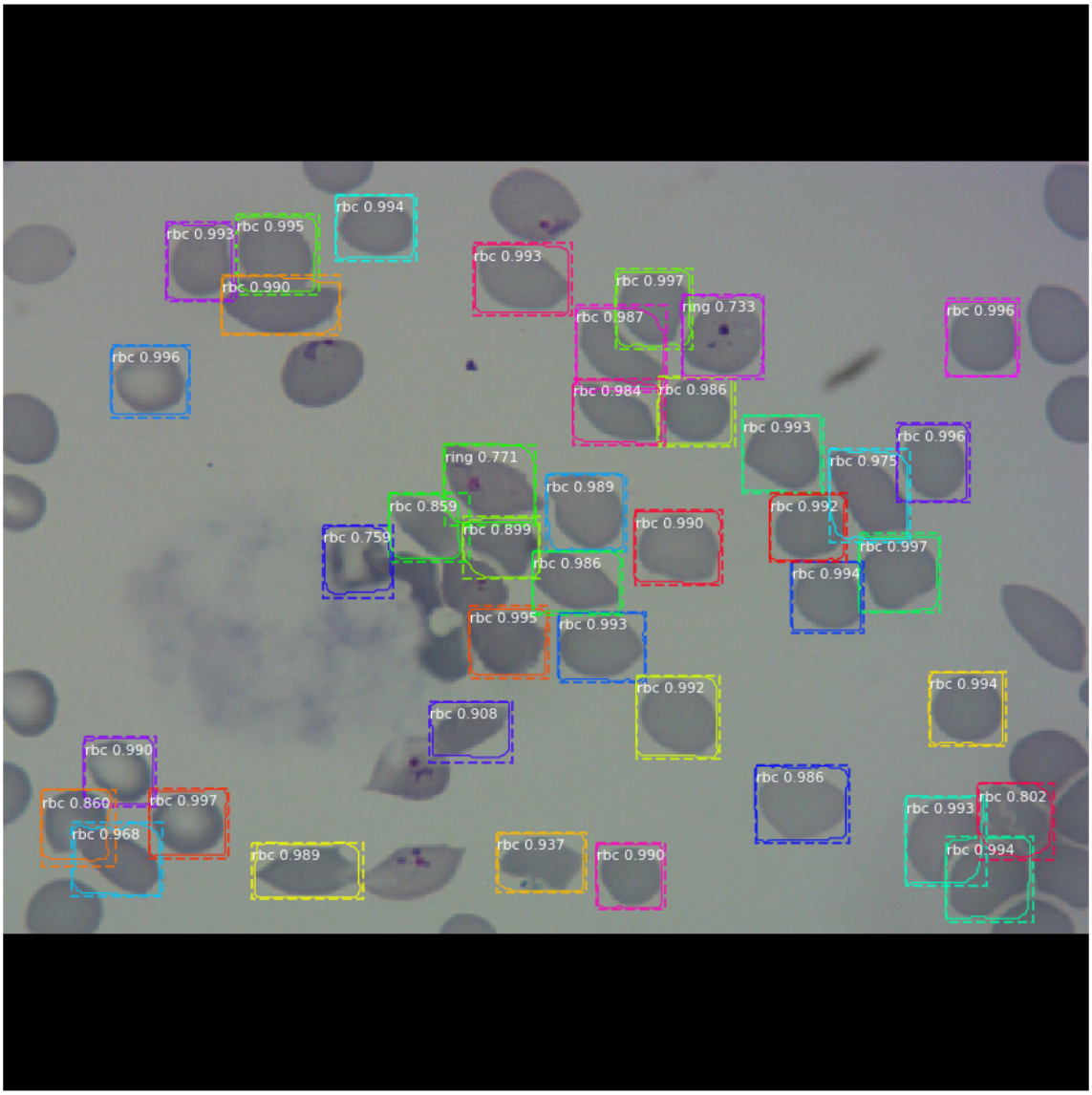
In [0]:
```
image_id = 16

print("\n Show Actual")
show_Actual(image_id)

print("\n Detecting RBC Cells")
detect_malaria(image_id)

print("\n Detecting Infected Cells")
detect_infected(image_id)

print("\n Detecting All")
detect_all(image_id)
```

Show Actual



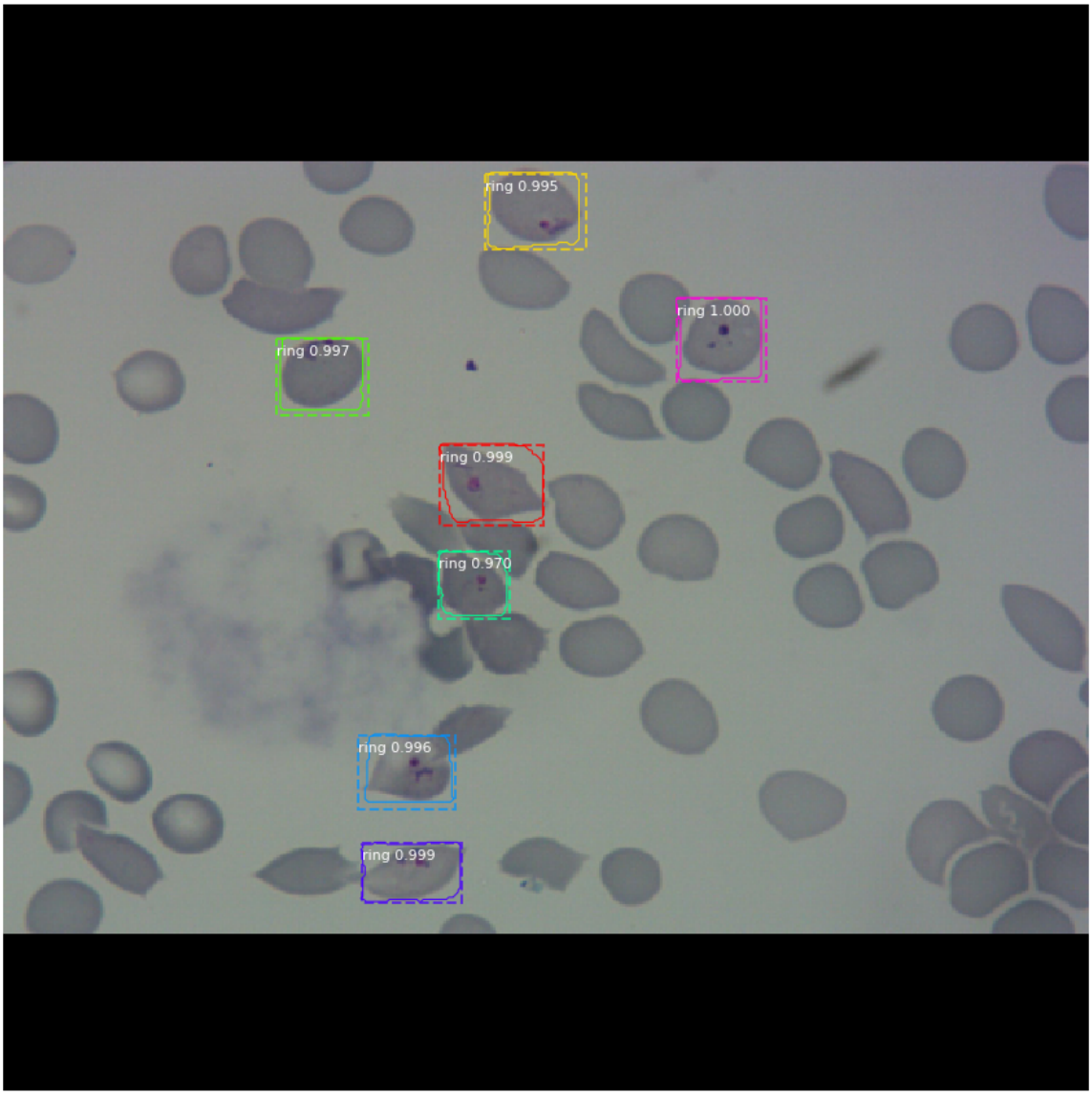

```
 Detecting RBC Cells
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
157.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
50.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
```

Predicted RBC

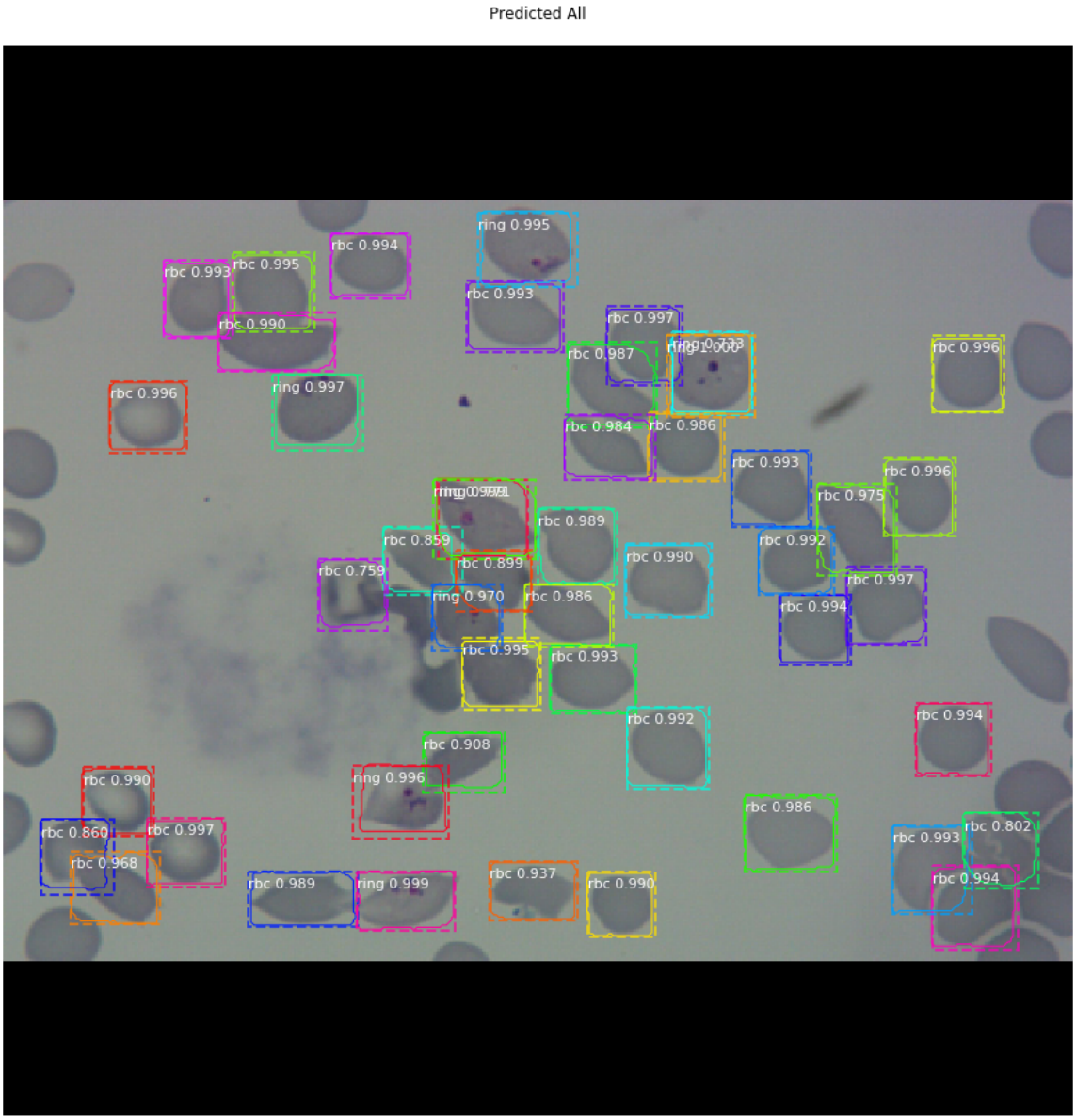



```
 Detecting Infected Cells
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
157.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
50.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
```

Predicted Infected



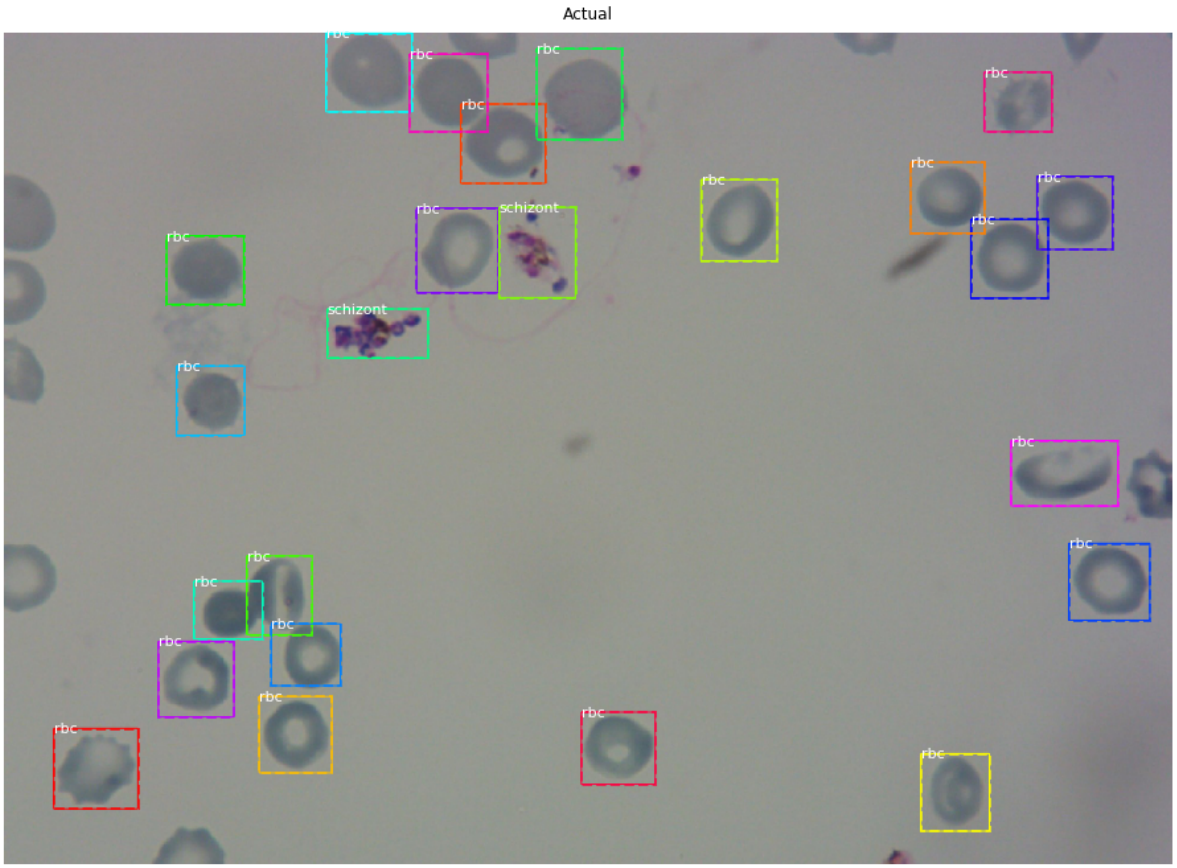

```
 Detecting All
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
157.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
50.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
157.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
50.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
```



Predicting 'schizont'

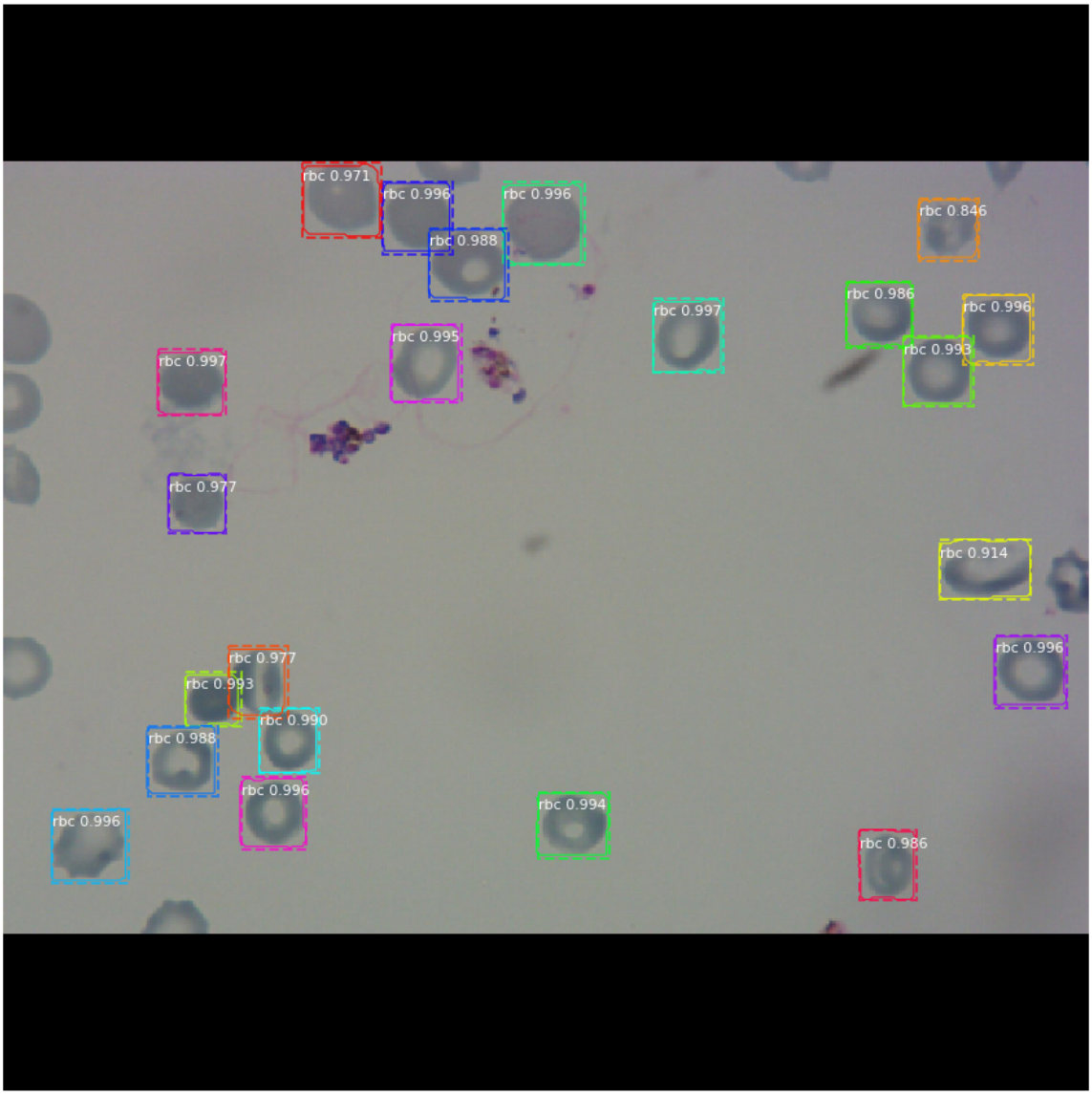
In [0]:
```
image_id = 251

print("\n Show Actual")
show_Actual(image_id)

print("\n Detecting RBC Cells")
detect_malaria(image_id)

print("\n Detecting Infected Cells")
detect_infected(image_id)

print("\n Detecting All")
detect_all(image_id)
```

Show Actual



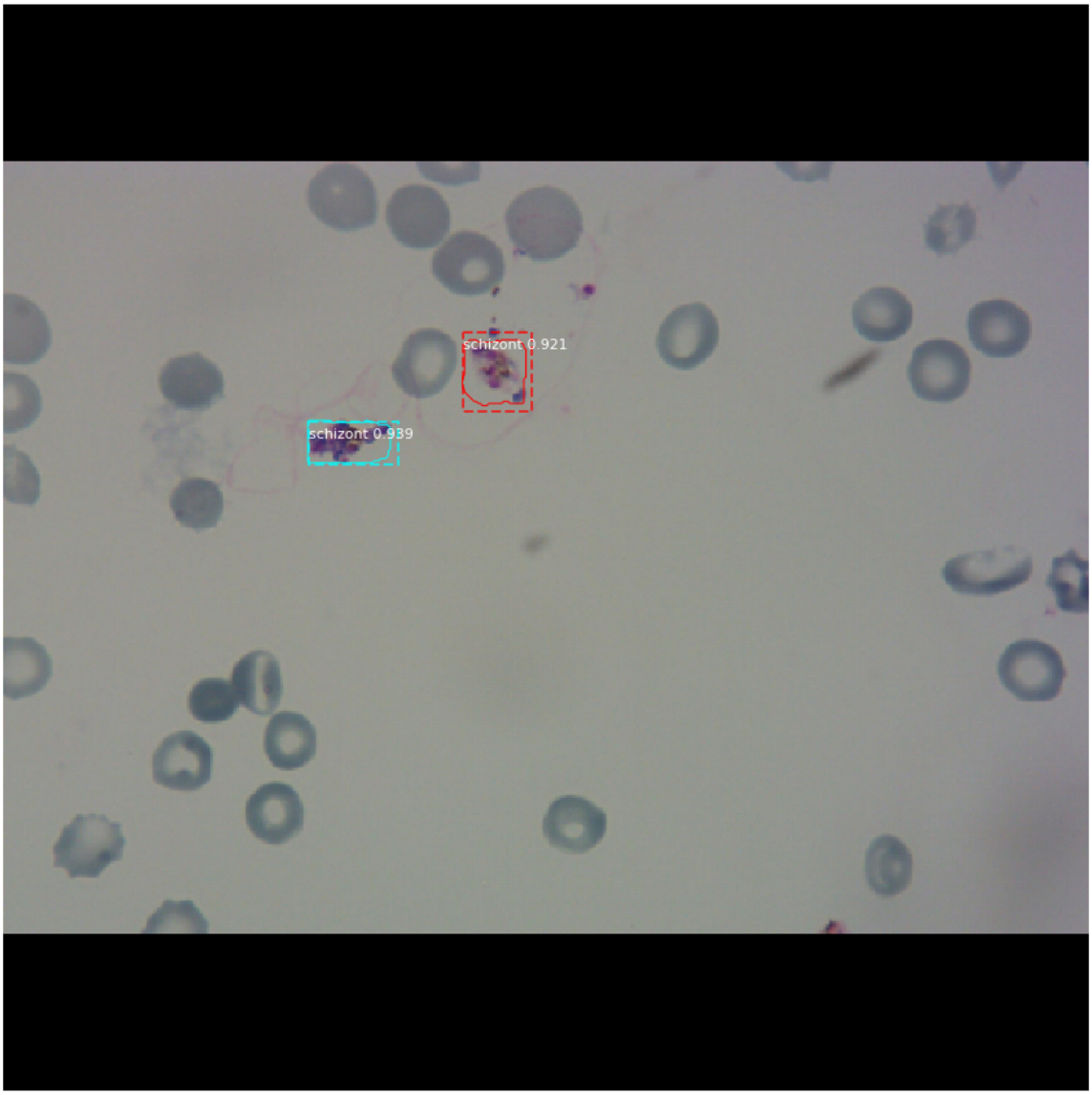

```
 Detecting RBC Cells
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
164.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
51.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
```

Predicted RBC

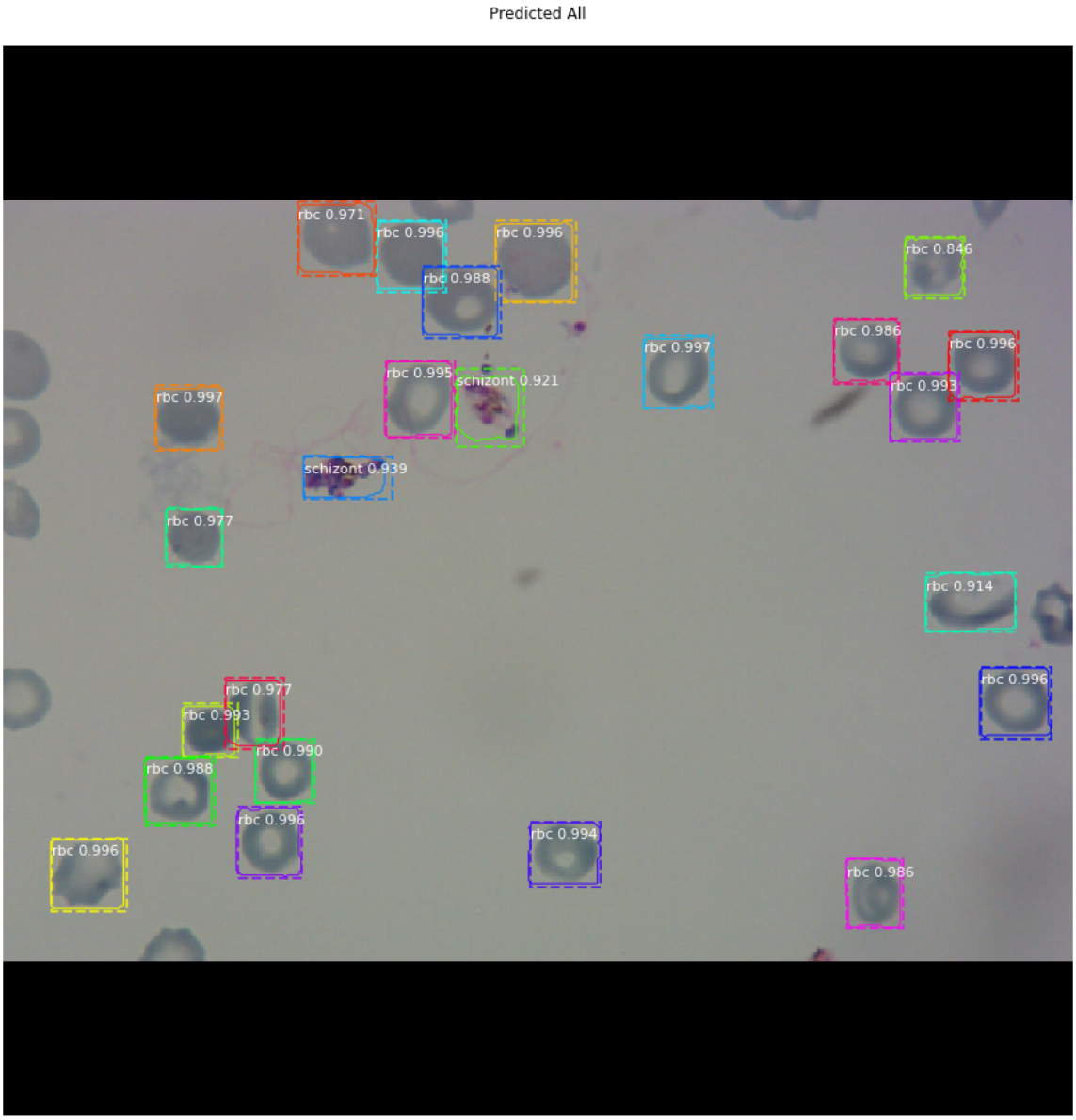



```
 Detecting Infected Cells
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
164.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
51.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
```

Predicted Infected



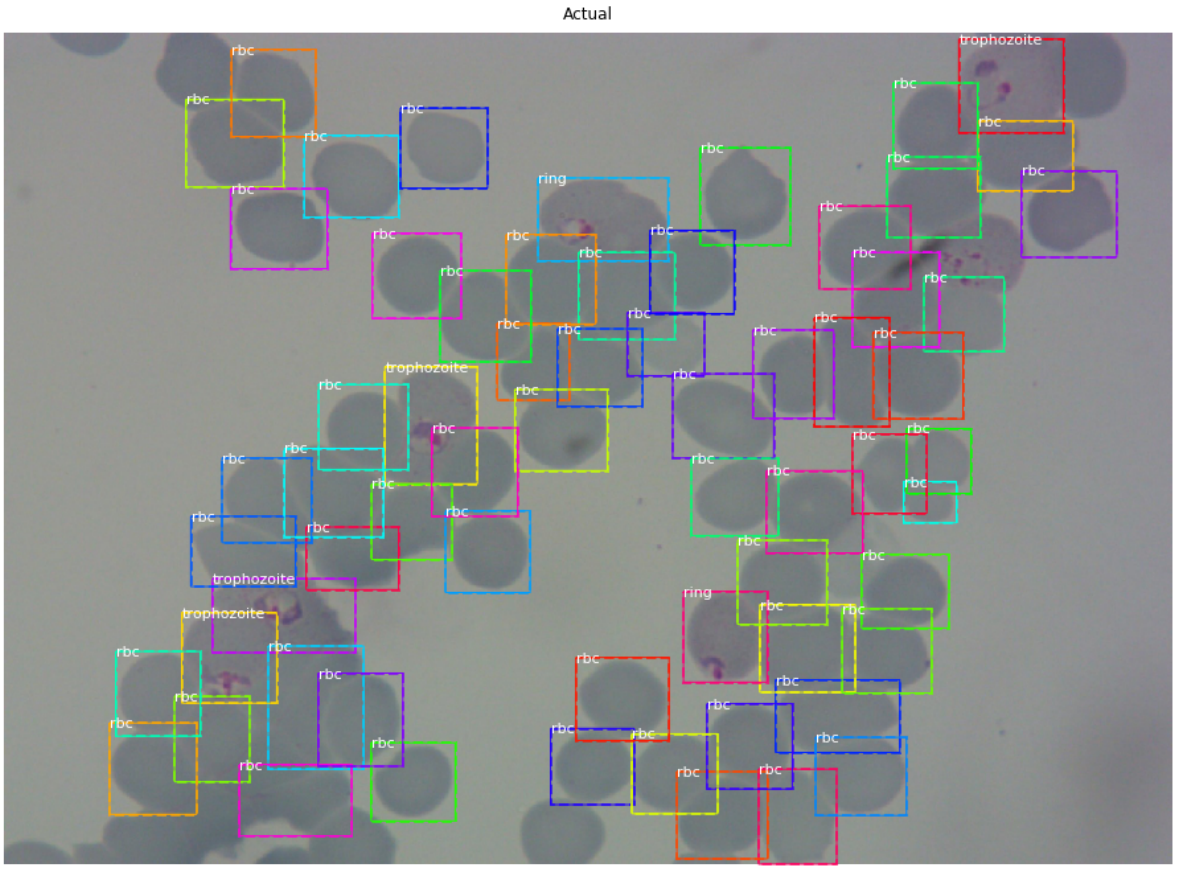

```
 Detecting All
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
164.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
51.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
164.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
51.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
```



Predicting 'multiple infections'

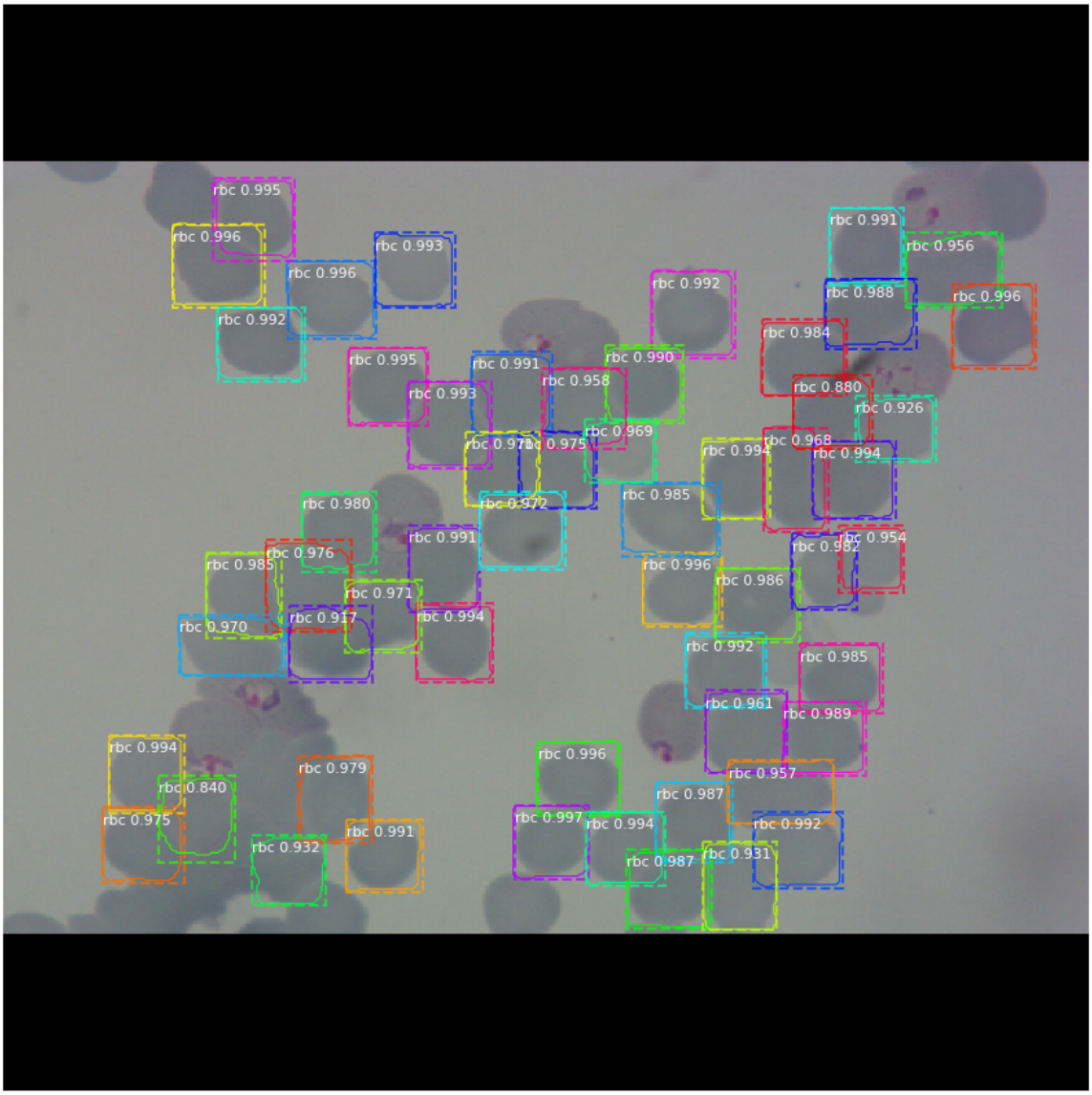
In [0]:
```
image_id = 43

print("\n Show Actual")
show_Actual(image_id)

print("\n Detecting RBC Cells")
detect_malaria(image_id)

print("\n Detecting Infected Cells")
detect_infected(image_id)

print("\n Detecting All")
detect_all(image_id)
```

Show Actual



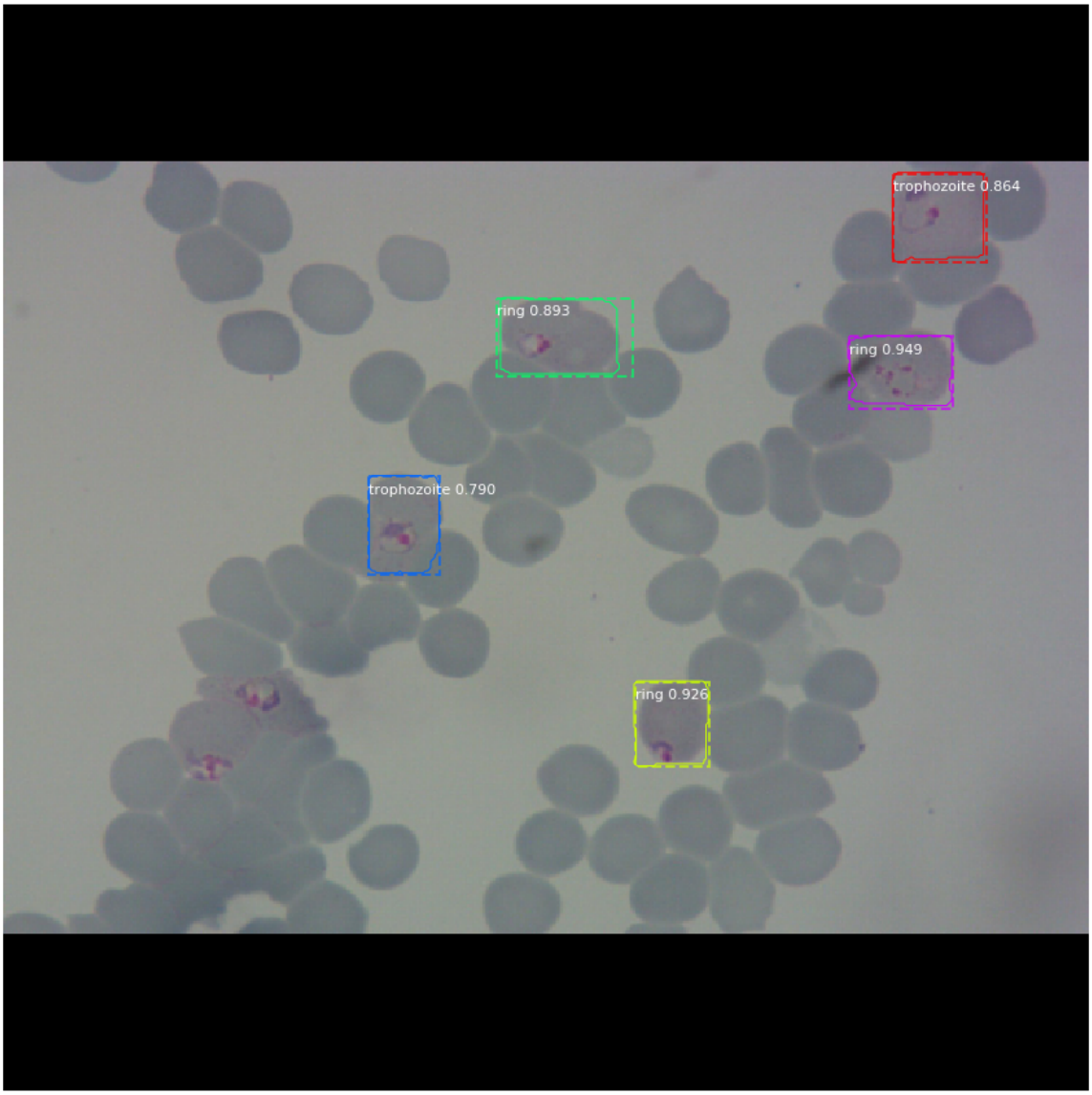

```
 Detecting RBC Cells
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
162.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
54.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
```

Predicted RBC



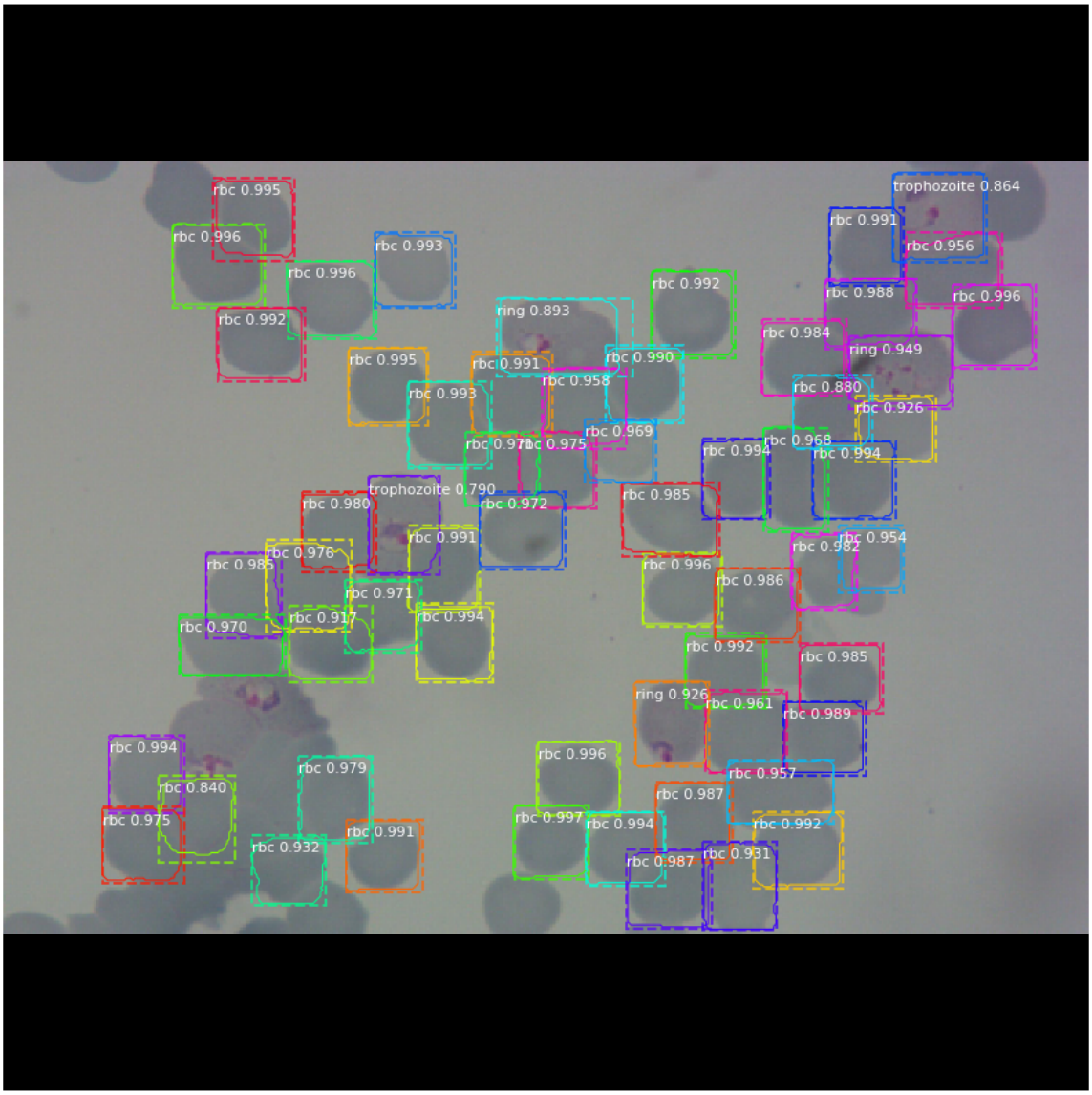

```
 Detecting Infected Cells
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
162.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
54.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
```

Predicted Infected




```
 Detecting All
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
162.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
54.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
Processing 1 images
image                    shape: (512, 512, 3)         min:    0.00000  max:
162.00000  int32
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:
54.10000  float64
image_metas              shape: (1, 20)               min:    0.00000  max:
512.00000  int64
anchors                  shape: (1, 65472, 4)         min:   -0.70849  max:
1.58325  float32
```

Predicted All



Conclusion :

1. ***In this Project we came accross a phenomenal Study of Computer Vision. How it can be used to predict Infected Malaria cells apart from non-Infected RBC, reducing Human's effort to detect them and resulting in better application for Health Science.***
2. ***We can see that there are still flaws in the Model, which can be enhanced in the coming future. It still lacks in predicting 'gametocyte' infection, we can make our model precise to a point where it can predict all of them without any loss.***
3. ***The train mAp : and test mAp : are quite good for the first version of the model.***
4. ***Also due to huge amount of 'rbc' domination the model could not be trained more on other categories.***
5. ***Mask R-CNN is applied on Malaria Data cells. It works fairly well on cells identification to Red Blood cells and Parasites infected cells.***

References :

https://www.analyticsvidhya.com/blog/2018/11/implementation-faster-r-cnn-python-object-detection/ (https://www.analyticsvidhya.com/blog/2018/11/implementation-faster-r-cnn-python-object-detection/)

https://towardsdatascience.com/faster-r-cnn-object-detection-implemented-by-keras-for-custom-data-from-googles-open-images-125f62b9141a (https://towardsdatascience.com/faster-r-cnn-object-detection-implemented-by-keras-for-custom-data-from-googles-open-images-125f62b9141a)

https://towardsdatascience.com/review-faster-r-cnn-object-detection-f5685cb30202 (https://towardsdatascience.com/review-faster-r-cnn-object-detection-f5685cb30202)

https://github.com/RockyXu66/Faster_RCNN_for_Open_Images_Dataset_Keras (https://github.com/RockyXu66/Faster_RCNN_for_Open_Images_Dataset_Keras)

https://github.com/kbardool/keras-frcnn (https://github.com/kbardool/keras-frcnn)

https://github.com/abc99lr/traffic-sign-faster-rcnn (https://github.com/abc99lr/traffic-sign-faster-rcnn)

https://medium.com/analytics-vidhya/computer-vision-techniques-implementing-mask-r-cnn-on-malaria-cells-data-e03b9fbeb6be (https://medium.com/analytics-vidhya/computer-vision-techniques-implementing-mask-r-cnn-on-malaria-cells-data-e03b9fbeb6be)

https://github.com/matterport/Mask_RCNN (https://github.com/matterport/Mask_RCNN)

https://github.com/keras-team/keras/tree/master/keras/applications (https://github.com/keras-team/keras/tree/master/keras/applications)

https://blog.insightdatascience.com/https-blog-insightdatascience-com-malaria-hero-a47d3d5fc4bb (https://blog.insightdatascience.com/https-blog-insightdatascience-com-malaria-hero-a47d3d5fc4bb)

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5840030/ (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5840030/)

https://machinelearningmastery.com/how-to-train-an-object-detection-model-with-keras/ (https://machinelearningmastery.com/how-to-train-an-object-detection-model-with-keras/)

https://stats.stackexchange.com/questions/21551/how-to-compute-precision-recall-for-multiclass-multilabel-classification (https://stats.stackexchange.com/questions/21551/how-to-compute-precision-recall-for-multiclass-multilabel-classification)