Object Detection Test

Alok Jadhav, Omkar Thawakar , SGGSIE&T Nanded.

Imports

In [1]:

```python
import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile

from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image



```

```
/Users/omkarchakradharthawakar/anaconda3/lib/python3.6/importlib/_bo
otstrap.py:219: RuntimeWarning: compiletime version 3.5 of module 't
ensorflow.python.framework.fast_tensor_util' does not match runtime
version 3.6
  return f(*args, **kwds)
```

Env setup

In [2]:

```python
# This is needed to display the images.
%matplotlib inline

# This is needed since the notebook is stored in the object_detect
sys.path.append("..")
```

Object detection imports

Here are the imports from the object detection module.

In [3]:

```python
from utils import label_map_util

from utils import visualization_utils as vis_util
```

Model preparation

Variables

Any model exported using the `export_inference_graph.py` tool can be loaded here simply by changing `PATH_TO_CKPT` to point to a new .pb file.

In [4]:
```
# What model to download.
MODEL_NAME = 'optic_disc_macula_graph'


# Path to frozen detection graph. This is the actual model that is
PATH_TO_CKPT = MODEL_NAME + '/frozen_inference_graph.pb'

# List of the strings that is used to add correct label for each b
PATH_TO_LABELS = os.path.join('training', 'optic_disc-detection.pb

NUM_CLASSES = 2
```

Load a (frozen) Tensorflow model into memory.

In [5]:
```
detection_graph = tf.Graph()
with detection_graph.as_default():
  od_graph_def = tf.GraphDef()
  with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
    serialized_graph = fid.read()
    od_graph_def.ParseFromString(serialized_graph)
    tf.import_graph_def(od_graph_def, name='')
```

Loading label map

Label maps map indices to category names, so that when our convolution network predicts `5`, we know that this corresponds to `airplane`. Here we use internal utility functions, but anything that returns a dictionary mapping integers to appropriate string labels would be fine

In [6]:
```
label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_
category_index = label_map_util.create_category_index(categories)
```

Helper code

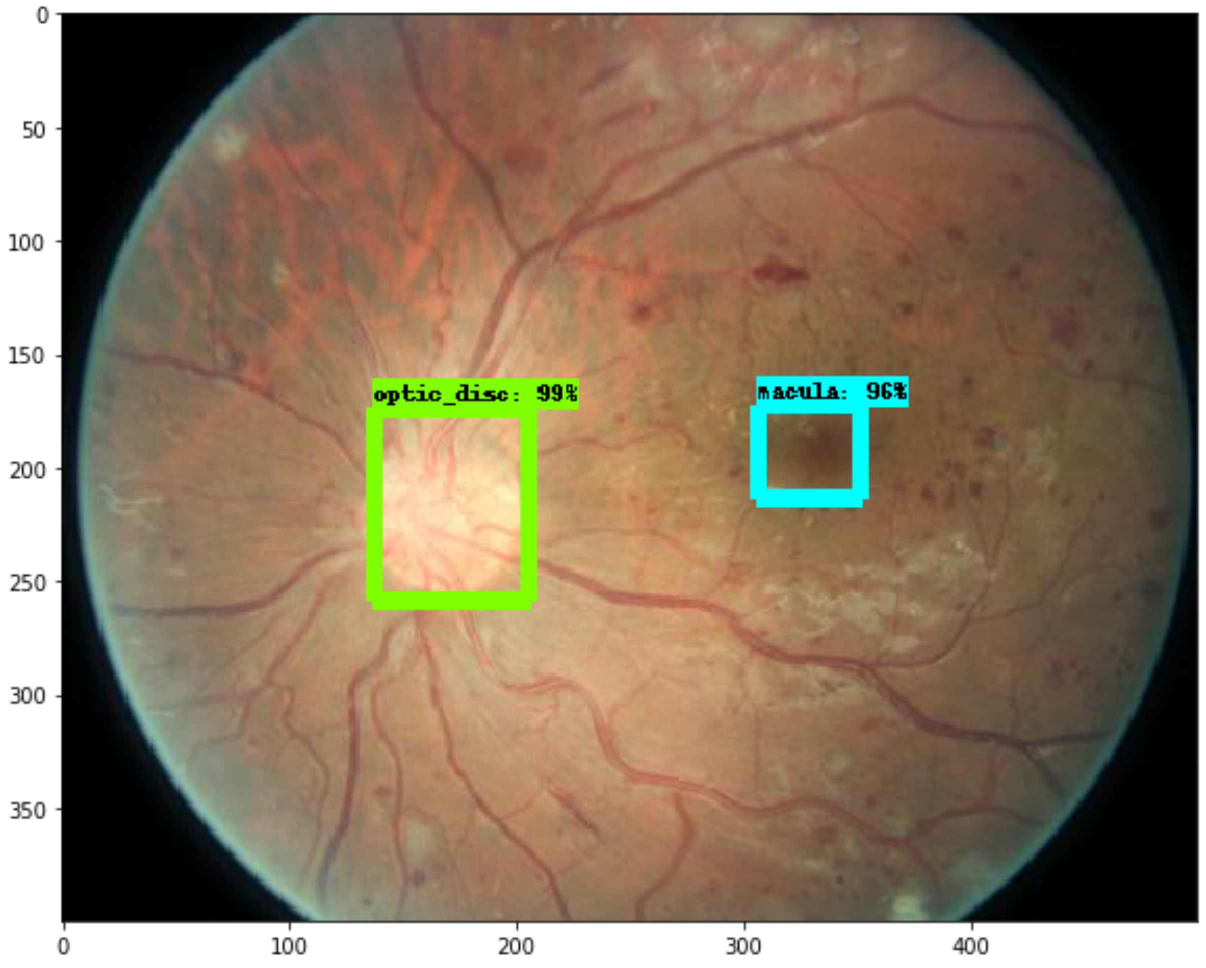
In [7]:
```
def load_image_into_numpy_array(image):
  (im_width, im_height) = image.size
  return np.array(image.getdata()).reshape(
      (im_height, im_width, 3)).astype(np.uint8)
```

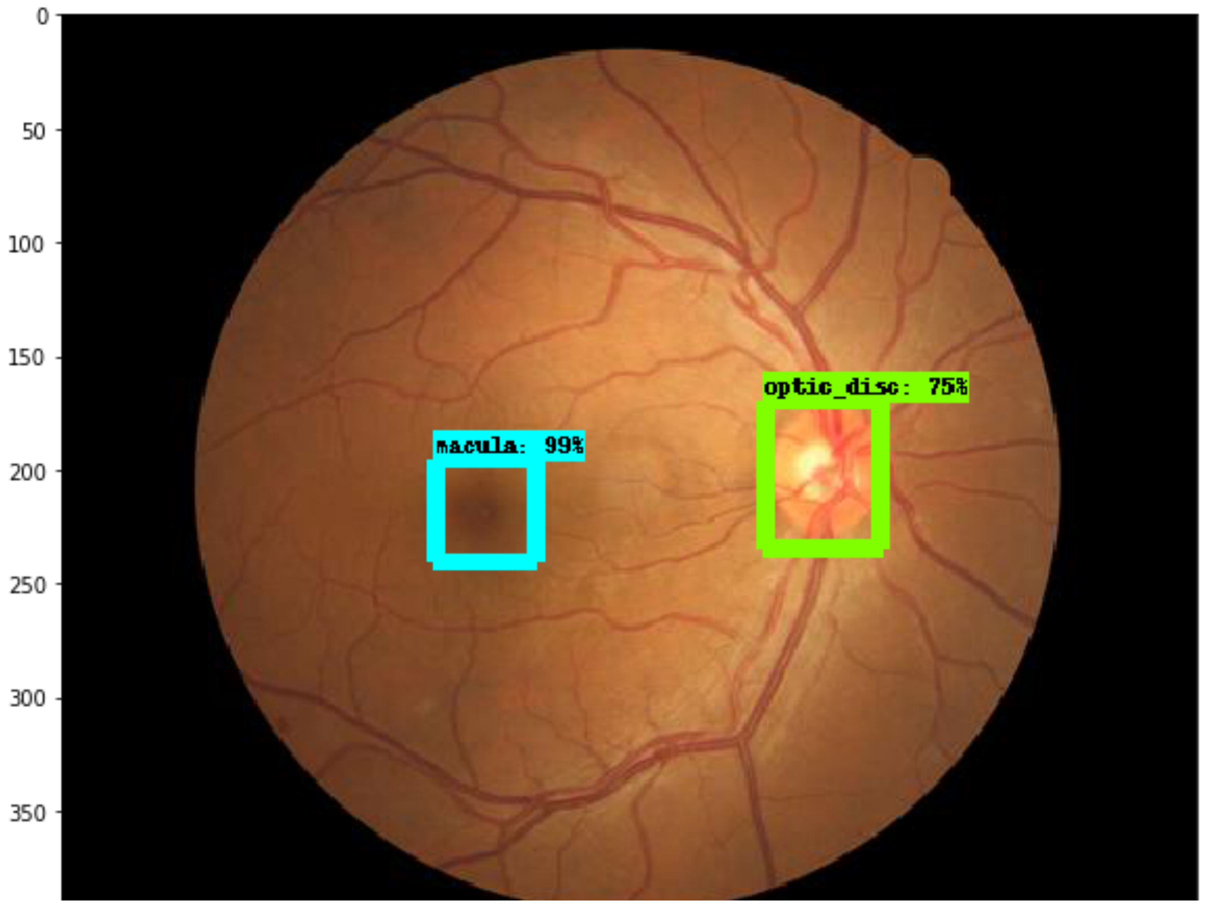
Detection

In [8]:
```
# For the sake of simplicity we will use only 2 images:
# image1.jpg
# image2.jpg
# If you want to test the code with your images, just add path to
PATH_TO_TEST_IMAGES_DIR = 'test_images'
TEST_IMAGE_PATHS = [ os.path.join(PATH_TO_TEST_IMAGES_DIR, 'image{

# Size, in inches, of the output images.
IMAGE_SIZE = (12, 8)
```

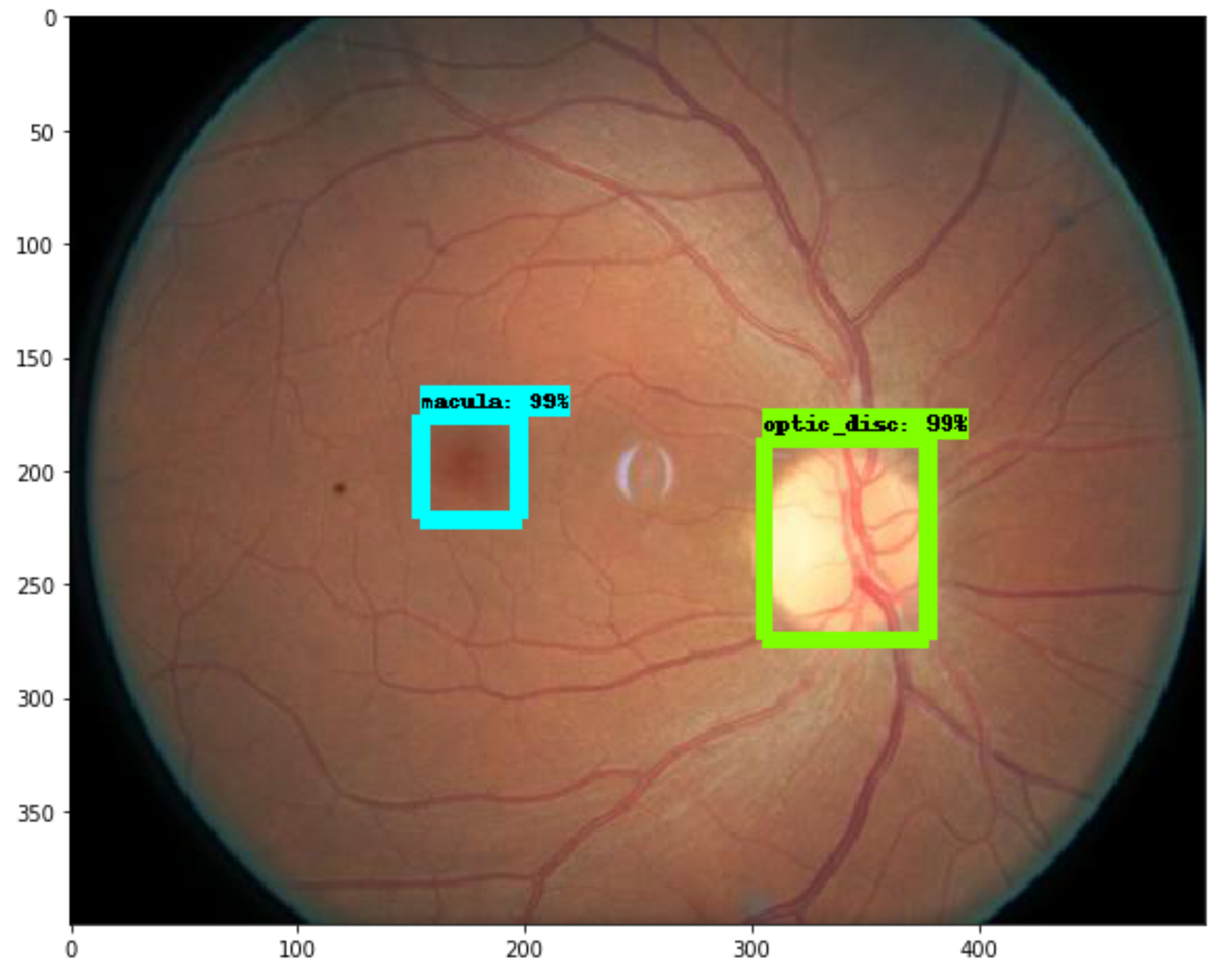
In [9]:
```
with detection_graph.as_default():
  with tf.Session(graph=detection_graph) as sess:
    # Definite input and output Tensors for detection_graph
    image_tensor = detection_graph.get_tensor_by_name('image_tenso
    # Each box represents a part of the image where a particular o
    detection_boxes = detection_graph.get_tensor_by_name('detectio
    # Each score represent how level of confidence for each of the
    # Score is shown on the result image, together with the class
    detection_scores = detection_graph.get_tensor_by_name('detecti
    detection_classes = detection_graph.get_tensor_by_name('detect
    num_detections = detection_graph.get_tensor_by_name('num_detec
    for image_path in TEST_IMAGE_PATHS:
      image = Image.open(image_path)
      # the array based representation of the image will be used l
      # result image with boxes and labels on it.
      image_np = load_image_into_numpy_array(image)
      # Expand dimensions since the model expects images to have s
      image_np_expanded = np.expand_dims(image_np, axis=0)
      # Actual detection.
      (boxes, scores, classes, num) = sess.run(
          [detection_boxes, detection_scores, detection_classes, n
          feed_dict={image_tensor: image_np_expanded})
      # Visualization of the results of a detection.
      vis_util.visualize_boxes_and_labels_on_image_array(
          image_np,
          np.squeeze(boxes),
          np.squeeze(classes).astype(np.int32),
          np.squeeze(scores),
          category_index,
          use_normalized_coordinates=True,
          line_thickness=8)

      plt.figure(figsize=IMAGE_SIZE)
```

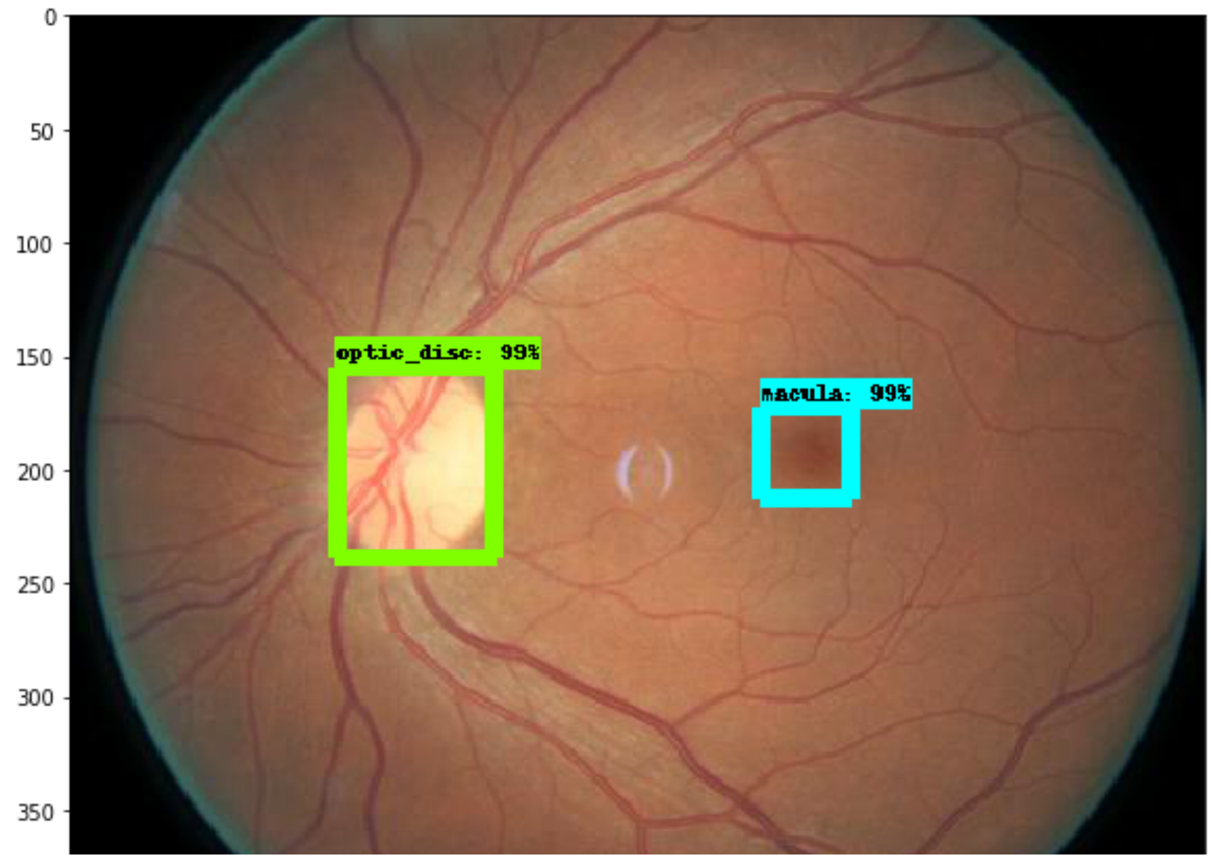
```
        plt.imshow(image_np)
```



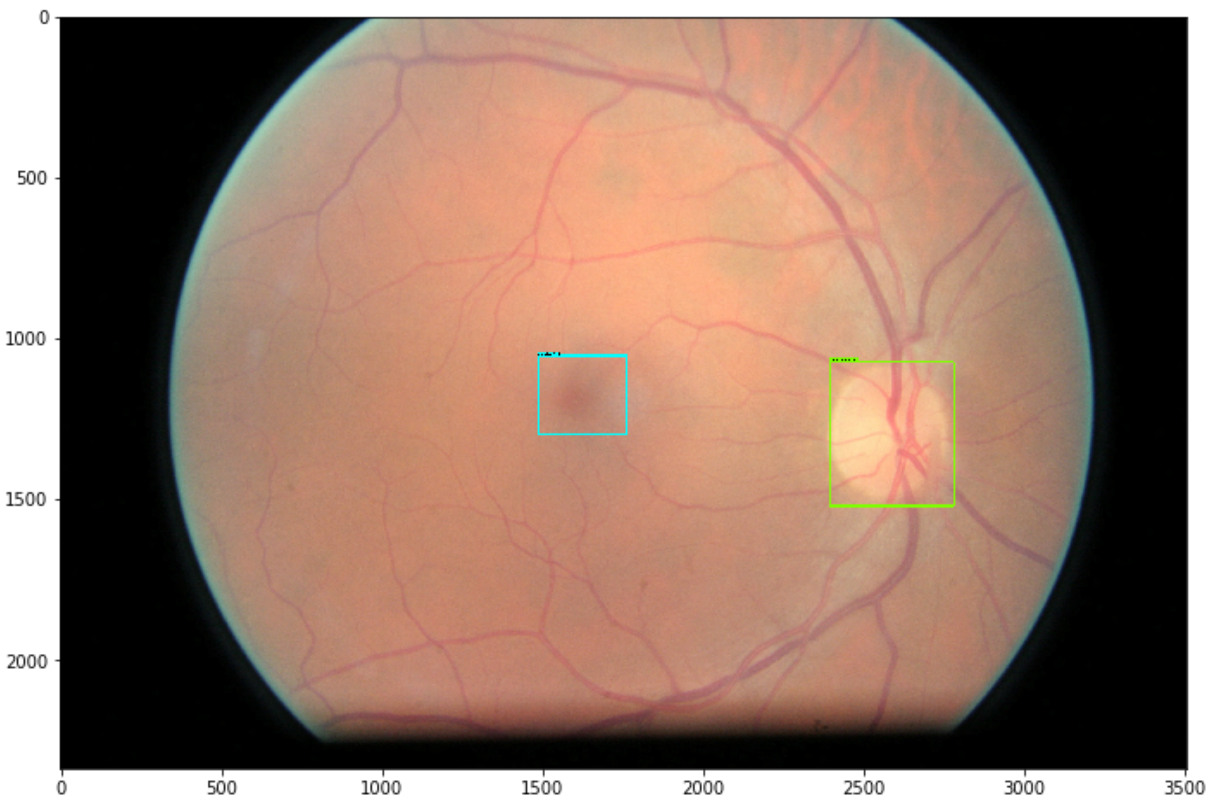


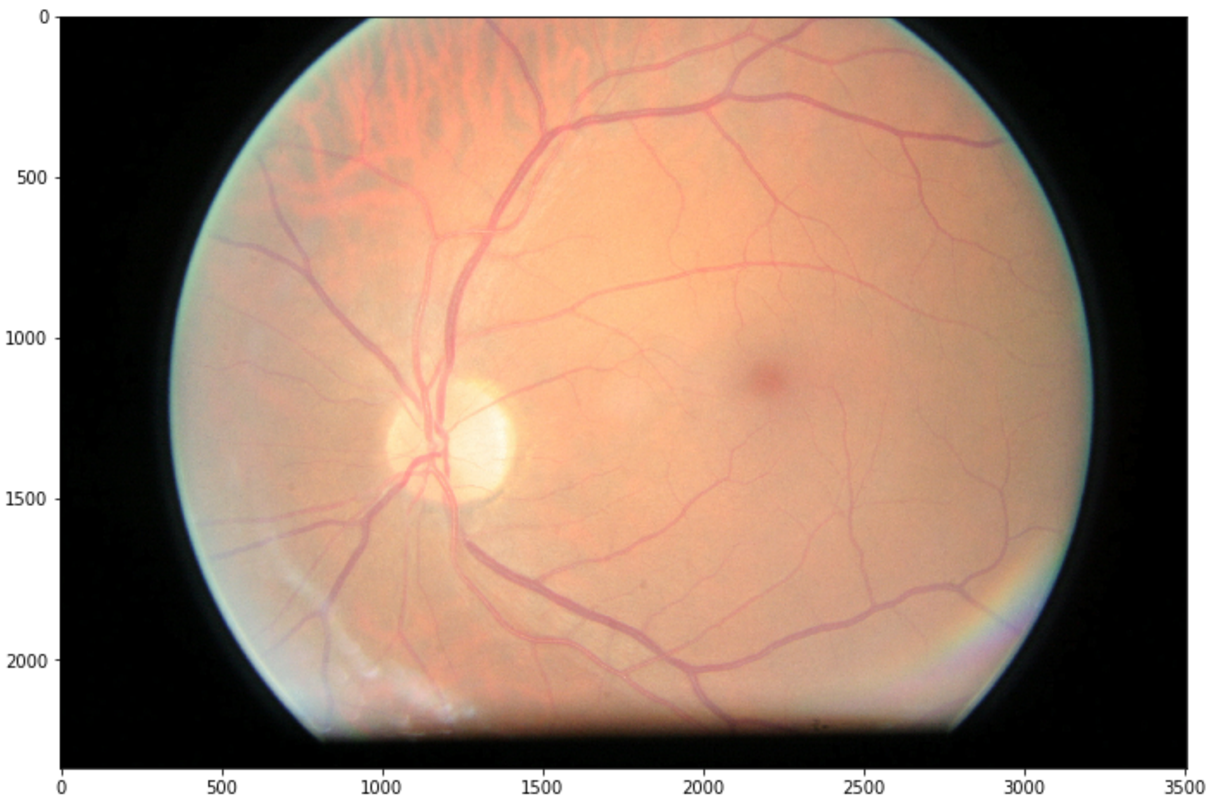
0
100
200
300
400

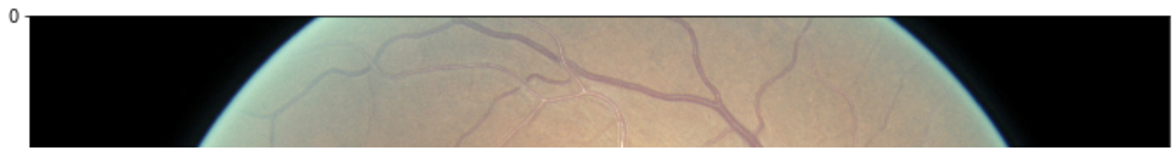

macula: 99%
optic_disc: 99%
0
50
100
150
200
250
300
350
0
100
200
300
400

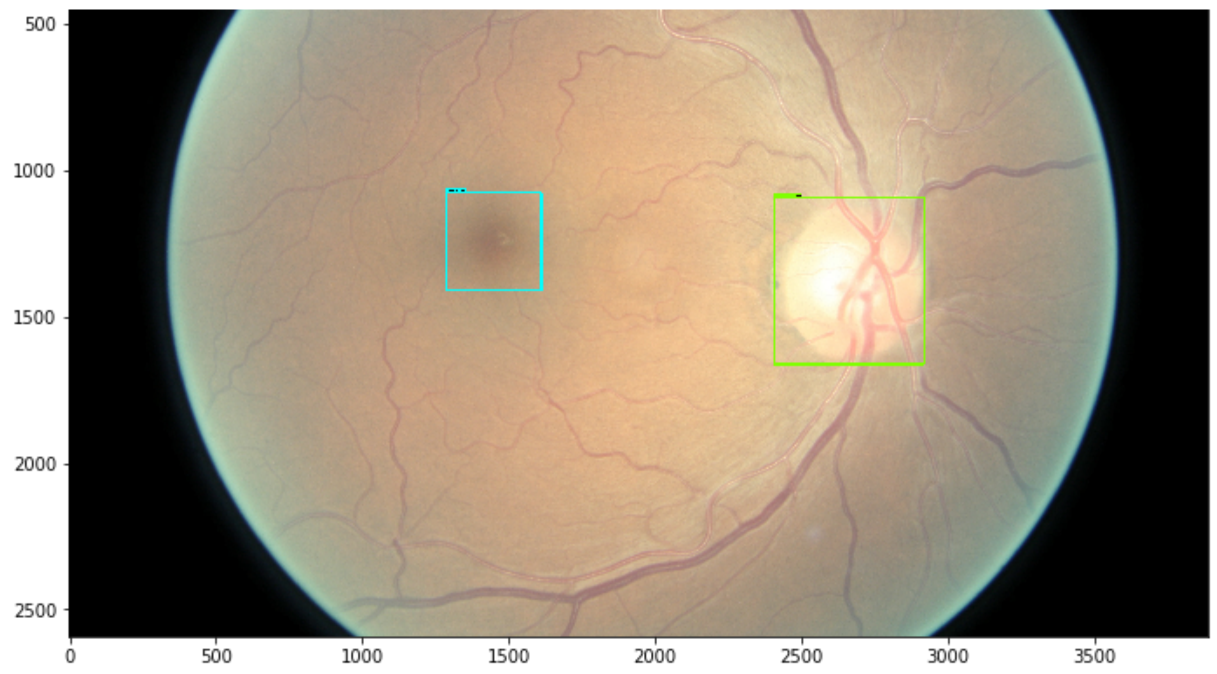

optic_disc: 99%
macula: 99%
0
50
100
150
200
250
300
350

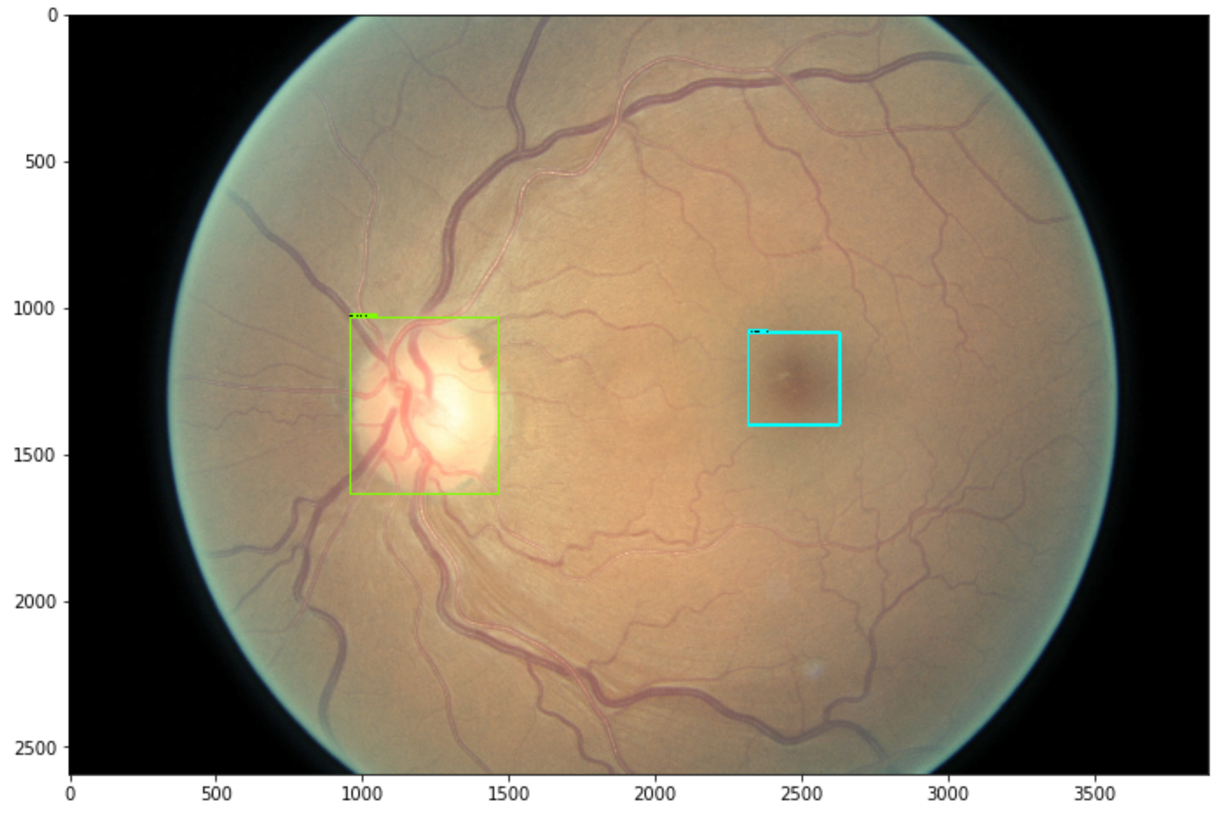
0
100
200
300
400

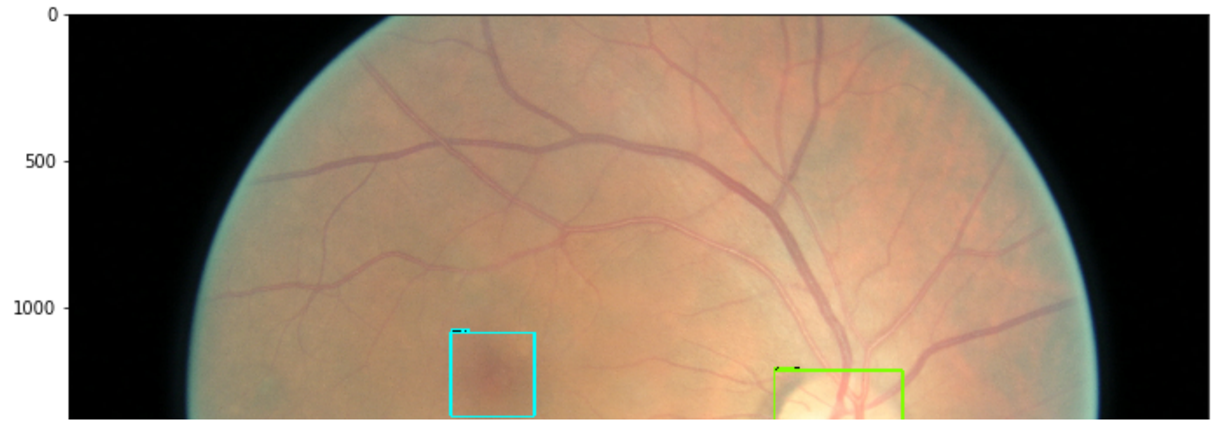

0
500
1000
1500
2000
0
500
1000
1500
2000
2500
3000
3500

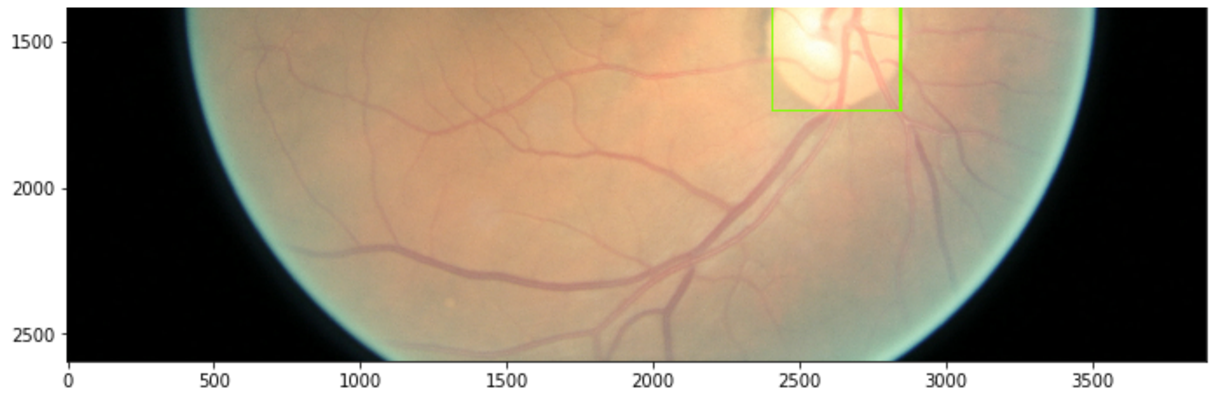

0
500
1000
1500
2000
0
500
1000
1500
2000
2500
3000
3500


0


500
1000
1500
2000
2500
0
500
1000
1500
2000
2500
3000
3500


0
500
1000
1500
2000
2500
0
500
1000
1500
2000
2500
3000
3500


0
500
1000



In [ ]:
```
1
```