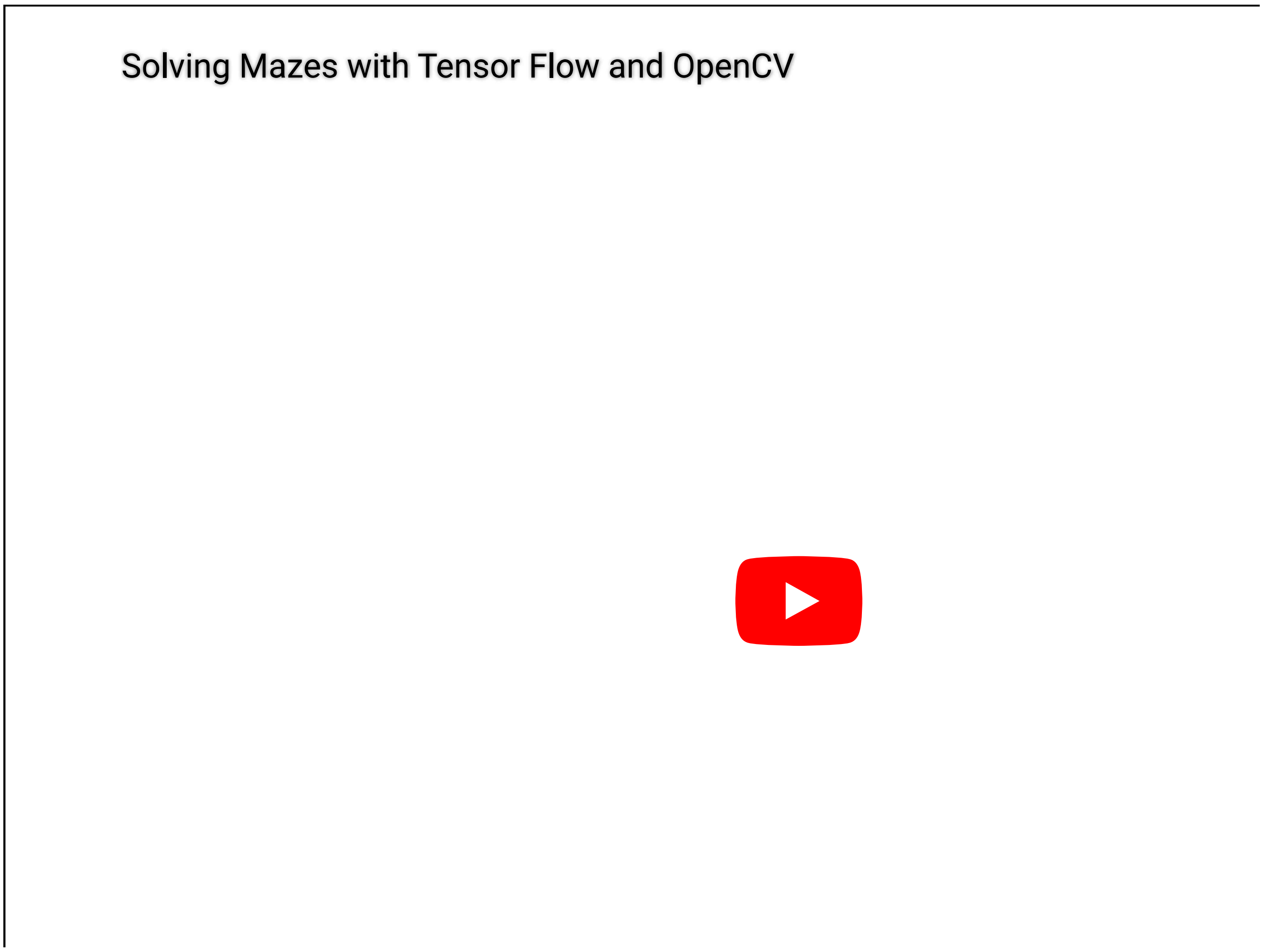
Manoochehr Akhlaghinia · Follow
4 min read · Apr 3, 2022

Listen Share

Solving Mazes with Tensor Flow and OpenCV

1. **Introduction**

Solving a maze with a camera has been my personal challenge recently. This article shows the steps I have taken to turn an idea to a functioning tool.



The main components are:

· **Dataset:** Few hundreds of maze images in different size, quality, color but all in rectangular shape

· **Maze Detector:** A Tensor Flow based object detection model trained with Transfer Learning approach. An existing model has been trained with limited number of images (~ 500 images)

· **Maze Solver:** can be done in 3 steps

o Converting a raw image into a format able to be solved by a maze search

o Solving the maze with an existing algorithm (Breath First Search)

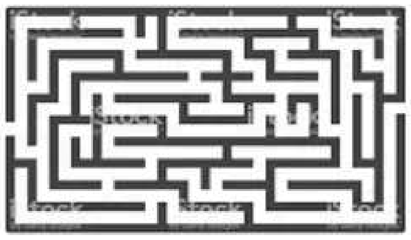
o Restore the image with solution overlaid

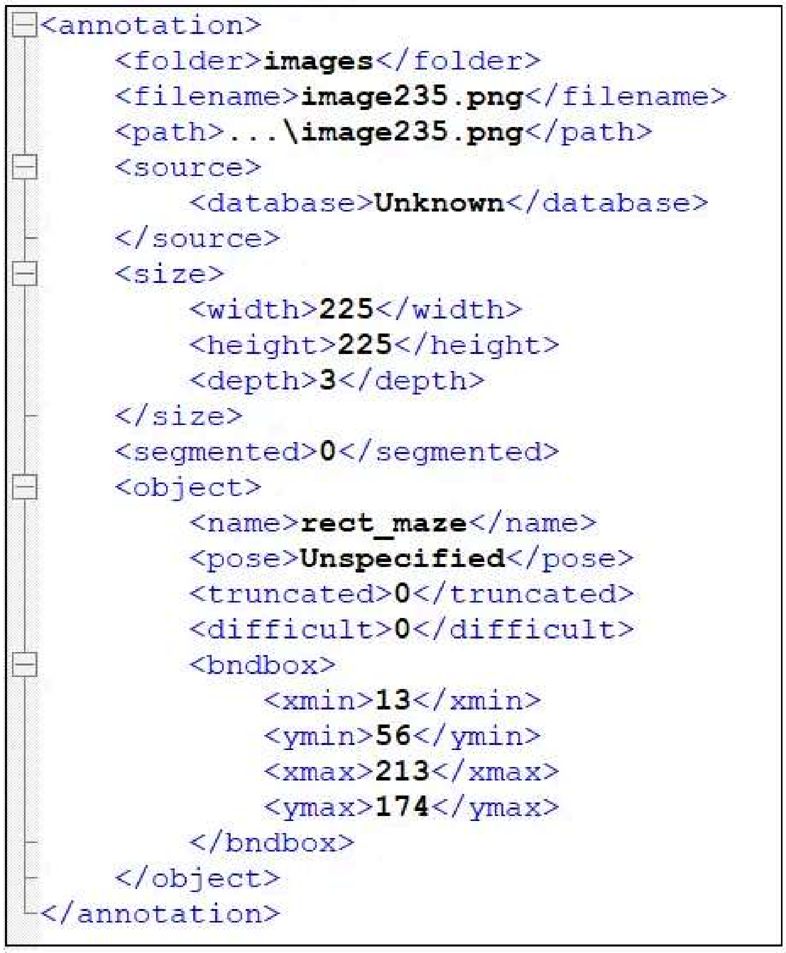
· **Main Framework**: A framework for activating camera and executing image detection model and maze solver

2. Dataset

Around 500 random rectangular mazes in different shape, size, and color have been used to train a maze detector model. An extension called Fatkun Batch Download Image can be utilized to download batches of images. Below picture shows few of the images in the dataset

Images are then needed to be properly labelled. There are several labeling tools available with different functionality but labelimg library is very handy. Labelimg library is an interface which allows labeling object(s) in each image (label name and position of the object). Below image shows an example from labelimg. The output xml file has the object/class name which is rect_maze and pixel coordinates of the maze in the image defined by (xmin, ymin) and (xmax, ymax) points/pixels.





```xml
<annotation>
    <folder>images</folder>
    <filename>image235.png</filename>
    <path>...\image235.png</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>225</width>
        <height>225</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>rect_maze</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>13</xmin>
            <ymin>56</ymin>
            <xmax>213</xmax>
            <ymax>174</ymax>
        </bndbox>
    </object>
</annotation>
```

Figure 1. Image labeling with labelimg.

3. Maze Detector

Building an object detector model from scratch is possible but not recommended as it is considerably a time-consuming process in terms of preparing dataset and model tuning. Transfer Learning (or simply taking advantage of existing trained models) allows achieving a good object detection model with a few hundred images.

A model from Tensor Flow Object Detection Model Zoo 1 and 2 can be, via re-training, re-purposed for maze detection. Each model in the model Zoo has its detection speed (in millisecond) and mean Average Precision score taken on COCO dataset. Generally, models with higher accuracy are relatively slower.

The process of re-training a coco model is out of the scope this article (detailed procedure can be found here), however a python script should be written to do below steps are as below:

· Importing Tensor Flow

· Loading a detection model

· Freezing the last layer of detection model

· Pointing model to a folder containing the training images

· Saving the trained model after reaching an acceptable accuracy

4. Maze Solver

The algorithm to solve an image giving a raw image is the most complex part. It has been done in below steps:

Step 1: Converting a raw image (detected by Maze Detector) into a format for a maze search algorithm. This has been performed in tow steps. First, to convert image to a 2D array with OpenCV. Second, to shrink the 2D array into a binary 2D array which 0 and 1 represent wall and way, respectively.

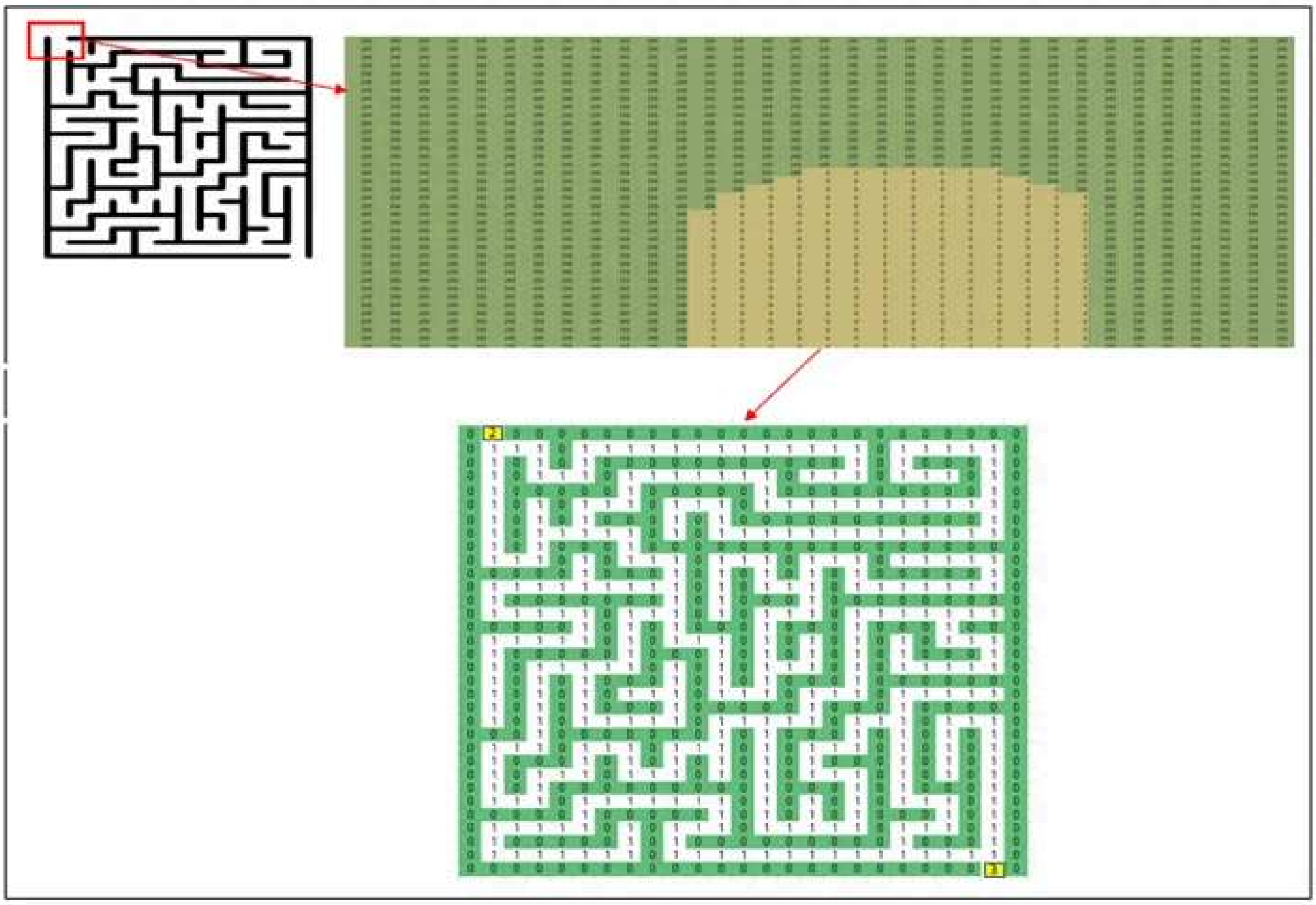


Figure 2. Image conversion.

Step 2: Once 2D array is ready, a maze solution algorithm can be applied to find a way from 2 to 3 through 1s (only up, down, left, and right allowed). A modified First Breath Search has been adapted in this work, Figure 3.

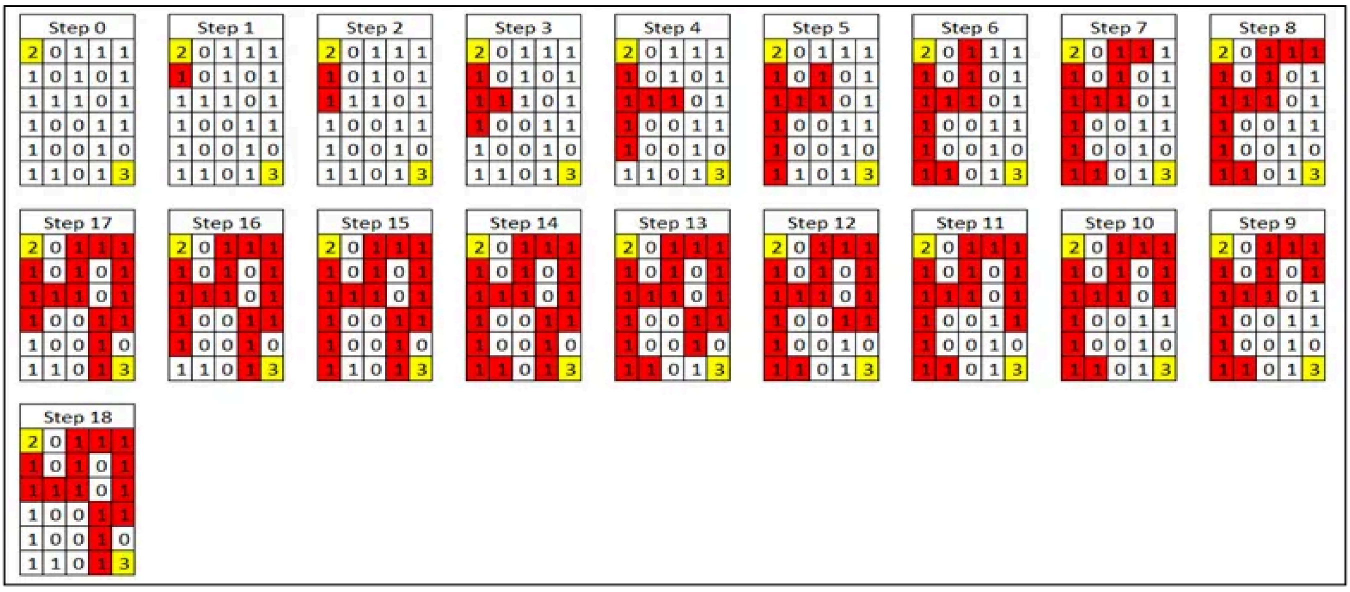


Figure 3. Modified First Breath Search Algorithm.

Step 3: Once maze is solved, original maze should be reconstructed with solution overlaid, Figure 4.

The maze solution can be found in this GitHub repository (link).

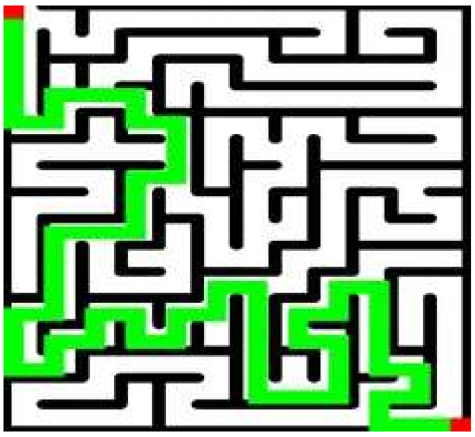


Figure 4. Resorting original maze with solution overlaid.

5. Main Framework

Finally, a script is required to activate a camera, capture the maze (in an image hold in front of camera) by detection model, run the solver and, finally, overlay the solution on the captured maze. Below script performs above steps.

```
import time
import numpy as np
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as vis_util
import cv2

cap = cv2.VideoCapture(0)

# importing maze solver package
from solve_maze import solve_maze_2

# Path to frozen detection graph. This is the actual model that is
used for the object detection.
PATH_TO_CKPT = '.../frozen_inference_graph.pb'

# path to a pbtxt file containing classes the objec was trained on.
PATH_TO_LABELS = 'object-detection.pbtxt'
NUM_CLASSES = 1

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='')

label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories =
label_map_util.convert_label_map_to_categories(label_map,
max_num_classes=NUM_CLASSES, use_display_name=True)
category_index = label_map_util.create_category_index(categories)

# threshold for the accuracy of detection. We dont want to run sover
on image with accuracy below this threshold
detection_threshold=0.995

with detection_graph.as_default():
    with tf.Session(graph=detection_graph) as sess:
        pause=30
        initial_flag=True
        while True:
            if initial_flag==True:
                ret, image_np = cap.read()
                ret, image_np_clean=cap.read()
                image_np_expanded = np.expand_dims(image_np, axis=0)
                image_tensor =
detection_graph.get_tensor_by_name('image_tensor:0')
                boxes =
detection_graph.get_tensor_by_name('detection_boxes:0')
                scores =
detection_graph.get_tensor_by_name('detection_scores:0')
                classes =
detection_graph.get_tensor_by_name('detection_classes:0')
                num_detections =
detection_graph.get_tensor_by_name('num_detections:0')
                (boxes, scores, classes, num_detections) = sess.run(
                    [boxes, scores, classes, num_detections],
                    feed_dict={image_tensor: image_np_expanded})
                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=2)
                if scores[0][0]>=detection_threshold:
                    pause=pause-1
                if (scores[0][0]>=detection_threshold)&(pause<0):
                    height, width, channels = image_np.shape
                    print("box : ", boxes[0][0])
                    detected_image_raw=image_np_clean[int(boxes[0]
[0][0]*height):int(boxes[0][0][2]*height),int(boxes[0][0]
[1]*width):int(boxes[0][0][3]*width),:]

cv2.imwrite('temp_image_2.tiff',detected_image_raw)


solution=solve_maze_2.master_solver(image='temp_image_2.tiff',
                                thershold_1=0.7,
                                thershold_2=0.5,
                                thershold_3=0.71)

                    x_offset=int(boxes[0][0][1]*width)
                    y_offset=int(boxes[0][0][0]*height)
                    image_np[y_offset:y_offset+solution.shape[0],
x_offset:x_offset+solution.shape[1]] = solution
                    cv2.imwrite('img2.png',image_np)

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=2)
                    time.sleep(5)
                    initial_flag=False

cv2.imshow('object detection', cv2.resize(image_np, (800, 600)))
            if cv2.waitKey(25) & 0xFF == ord('q'):
                cv2.destroyAllWindows()
                break
```

Maze TensorFlow Object Detection Maze Search Algorithms Opencv