Filtering and Visualizing the Detections

Now that we have got an idea about mask processing from the previous slide, let us take a closer look at the code by understanding each small snippet:

1.

for i in range(0, boxes.shape[2]): #For each detection

classID = int(boxes[0, 0, i, 1]) #Class ID

confidence = boxes[0, 0, i, 2] #Confidence scores

if confidence > threshold:

(H, W) = img.shape[:2]

box = boxes[0, 0, i, 3:7] \* np.array([W, H, W, H]) #Bounding box

(startX, startY, endX, endY) = box.astype("int")

boxW = endX - startX

boxH = endY - startY

* We shall iterate through each detection using for i in range(0, boxes.shape[2]).
* For each detection, we retrieve the classID and confidence score of the detection.
* If the confidence value of the detection is greater than the threshold, we consider it as a valid detection and further proceed to create the mask using the correspong mask polygon for this detection, as follows:
  + Get the height H and width W of the img.
  + Get the bounding boxes of this valid detection using boxes[0, 0, i, 3:7] and normalize the bounding boxes using boxes[0, 0, i, 3:7] \* np.array([W, H, W, H]). Thus the normalized bounding boxes of the currect valid detection is stored in box.
  + Get the width boxW and height boxH of the bounding box.

2.

mask = masks\_polygons[i, classID]

plt.imshow(mask)

plt.show()

print("Shape of individual mask", mask.shape)

* Now, after getting the normalized bounding box coordinates of the current valid detection, we would extract the pixel-wise segmentation for that detection using mask = masks\_polygons[i, classID].
* We then visualize the mask and print the shape of it.

3.

mask = cv2.resize(mask, (boxW, boxH), interpolation=cv2.INTER\_CUBIC)

print ("Mask after resize", mask.shape)

mask = (mask > threshold)

* We shall then resize this 15 x 15 mask to the dimensions of the bounding box (boxW, boxH).
* We then print the shape of the mask after resizing it.
* Then we convert the pixels of the mask to be in binary format using mask = (mask > threshold).

4.

roi = img[startY:endY, startX:endX][mask]

* Now, we extract the region of interest (ROI) from the image. It is the overlapping area of the mask in the img.

5.

color = COLORS[classID]

blended = ((0.4 \* color) + (0.6 \* roi)).astype("uint8")

img[startY:endY, startX:endX][mask] = blended

* Now we get the blended form of the color to mark the ROI using this color. This blended color is nothing but the combination of the actual object roi and the random color color we have generated for each classID.
* Next, by using img[startY:endY, startX:endX][mask] = blended, we impart this blended color on the ROI of the image, thus forming the view of the object being overlapped with the color of ClassID.

6.

color = COLORS[classID]

color = [int(c) for c in color]

print (LABELS[classID], color)

cv2.rectangle(img, (startX, startY), (endX, endY), color, 2)

text = "{}: {:.4f}".format(LABELS[classID], confidence)

cv2.putText(img, text, (startX, startY - 5), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, color, 2)

* We finally draw the bounding boxes on the img, mark the detect labels along with their corresponding confidence values:
  + We first get the color corresponding to the ClassID.
  + Since each color is a representation of 3 values corresponding to R, G, B, we traverse through each of those values and convert them into integer format.
  + We then draw a rectangle using cv2.rectangle using the coordinate values startX, startY, endX, endY of the bounding boxes of the valid detections.
  + We then create the text with classID and the confidence value of the detection and write that text on the image using cv2.putText.