ECOINF-D-24-02221

Comments for the author

General comments

This paper describes SegColR, an R package that uses two deep learning models to enable semantic segmentation of biological images for e.g. background masking, and provides some code for performing color clustering on the resulting segmentations. I think the general idea of the package--a simple interface for zero-shot segmentation that allows users to provide their own labels and save the segmentation results--is incredibly useful, and can be applied to a huge range of problems in biological image analysis. Background masking, for example, is easily one of the most tedious tasks that most biologists encounter when trying to quantify images. I look forward to using a version of the package in my own research.

However, I think the package/paper as written have some technical and methodological issues that need to be resolved in order for this method to be used widely. I detail my recommendations below, but in summary: 1) I would remove the color clustering function and instead add options for batch processing and exporting to different formats; 2) the existing segmentation code needs to be carefully debugged since many of the examples do not work as written; and 3) the utility of the method should be demonstrated on a small example dataset. These changes would make the package much more usable to a wider audience of biologists, and would allow it to address some important and long-standing bottlenecks in image analysis.

Major recommendations

1. Debug the existing code

I tried out the R package version of SegColR, and I had a colleague who also has extensive computational image processing experience try out the Python scripts. We both ran into issues with installation, and almost every line of the R code as written in the examples and on the GitHub directory throws an error or a warning. Of course occasional warnings/errors are inevitable, but the example code should work, especially if the audience is biologists without extensive coding experience.

Installation

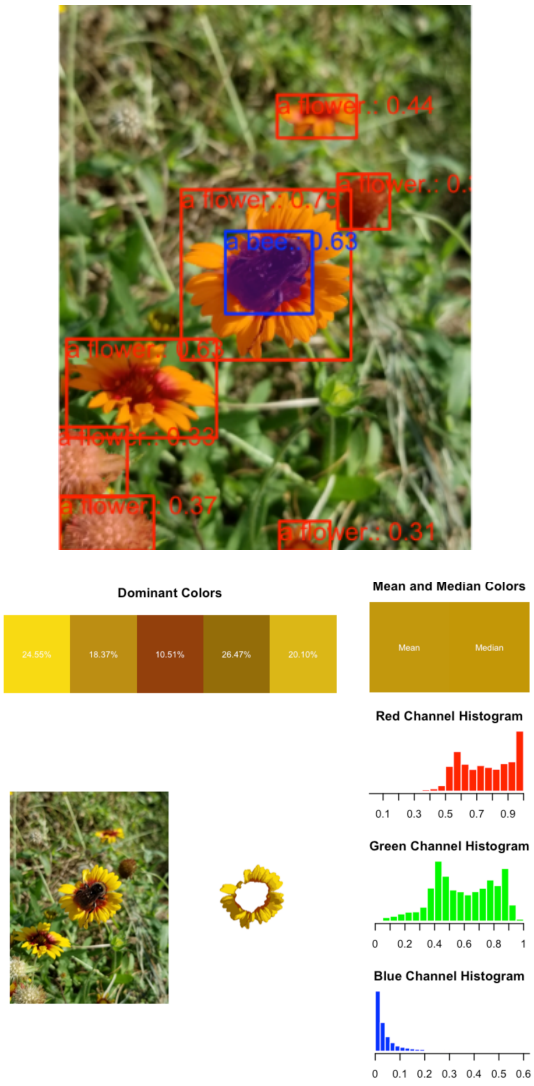
- In R: I already had Anaconda installed, but when I ran `setup_conda_environment()`, the function looked only for a Miniconda install. I ended up install Miniconda as well (which did work after cleaning up some of the failed conda environments), but either tell users to install Miniconda or test this out on some other machines with Anaconda installed.
- In Python: My colleague installed the Python environment using the provided .yaml file, but had to install some additional packages as well before the environment built. He recommends restricting the .yaml file for Python users to the minimal packages needed, and perhaps mentioning that since only these few packages are needed, it's not essential to use the provided .yaml file to build that specific environment (e.g. could just direct users to set up a conda environment with packages X, Y, and Z, since most of us have done this).

This is especially important for the R package version of the software, since the author specifies that this package is intended for people without much coding experience. I usually see users without much coding experience give up when this level of troubleshooting is required. A perfectly smooth installation process is not required (and probably impossible given the Python packages involved), but it could be made much smoother than this, and there should be more instructions provided.

Usage

The code in the paper is outdated (presumably the package structure was updated in between the paper submission and this review).

- For example, the first `process_masks_and_extract_colors()` usage (for the grouper image) throws an error because `ground_results` is now just a list of the command input/output and relevant filepaths for a segmentation call, and the provided error message ("No masks meet the score threshold") is not informative. This works fine on the GitHub README though, so just update things like this.
- I could not get the flower/bee example to work as written. The first step is fine (see image), except that for some reason the package appends periods to the ends of all my labels (I provided the labels as just "a flower" and "a bee"--if the periods are required for the labeling to work it should be stated). But providing a score threshold of 0.5 for the color segmentation only resulted in the larger flower being displayed.



This was the code used to produce the lower plot:

```
img <- system.file("extdata/images/AmericanBumbleBee.jpg", package = "SegColR")

ground_results <- grounded_segmentation_cli(
  image_path = img,
```

```
  labels = c("a bee", "a flower"),
  output_json = "./json/",
  output_plot = "./output/")

seg_results <- load_segmentation_results(ground_results$image_path,
                                         ground_results$json_path)

color_results <- process_masks_and_extract_colors(
  image = seg_results$image,
  masks = seg_results$mask,
  scores = seg_results$score,
  labels = c("a flower.", "a bee."), # had to append periods to the labels
  include_labels = c("a flower."),
  exclude_labels = c("a bee."),
  score_threshold = 0.5, # the flower with confidence of 0.63 (as shown in Fig. 3C) does n
ot appear
  n_colors = 5
)
```

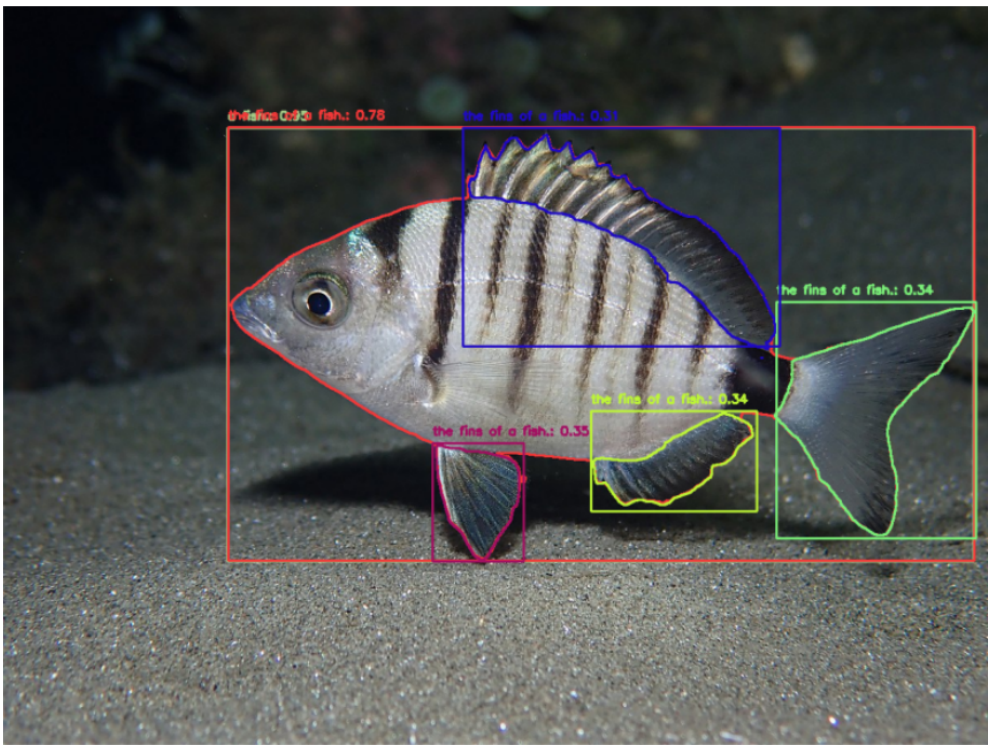
Oddly, when I provided the labels as just "bee" and "flower", the bee *was* included in the color analysis (although the confidence scores remained very similar), although the flower with a confidence of 0.63 still did not appear:

Not sure the underlying cause, but this is unexpected behavior. I do suggest below to remove the color segmentation component of the package, so this itself doesn't necessarily need to get fixed if the author takes my suggestion there. But it would be good to, for example, only export instances above a certain confidence threshold, which would draw on similar code.

- Example 3 threw a different error. This image was not provided on the GitHub, so I used a similar image (found via reverse image search) of a *Diplodus* bream from Wikimedia (https://commons.wikimedia.org/wiki/Category:Diplodus_puntazzo#/media/File:Diplodus_puntazzo_87357945.jpg). The original labeling worked quite well, but the color segmentation threw an error:

```
img <- "Diplodus_puntazzo_87357945.jpg"

# works very nicely except that labels was misspelled:
ground_results <- grounded_segmentation_cli(
  image_path = img,
  # labebls = c("a fish.", "the fins of a fish."),
  labels = c("a fish.", "the fins of a fish."),
  output_json = "json/",
  output_plot = "plot/")
```




```
> seg_results <- load_segmentation_results(ground_results$image_path,
                                           ground_results$json_path)

> # this doesn't work
> color_results <- process_masks_and_extract_colors(
+   image = seg_results$image,
+   masks = seg_results$mask,
+   scores = seg_results$score,
+   labels = c("a fish.", "the fins of a fish."),
+   include_labels = c("a fish."),
+   exclude_labels = c("the fins of a fish."),
+   score_threshold = 0.5,
+   n_colors = 3,
+ )
Error in sample.int(m, k) :
  cannot take a sample larger than the population when 'replace = FALSE'
```

I'm guessing from the error this is something to do with subsampling for the k-means clustering, but I didn't dig through the code to find out.

2. Remove the color clustering and expand the export options

The segmentation features of the package have the potential to be enormously useful, but the color clustering function seems ancillary, and I don't think there is a real use case for it that isn't covered already by dedicated software with more options. I recommend removing the color clustering function and instead improving the options surrounding the segmentations, namely, make it easier to batch process images and provide more export options.

The author already cites a few other packages that could be used in downstream analysis, e.g. patternize, pavo, recolorize, and micaToolbox/QCPA. All of these could take the results of SegColR as input, since they rely on (or can

use) segmented images. For example, the colordistance package (not cited here but seems relevant) uses background-masked images to perform color clustering of RGB images, similar to the color clustering function in SegColR. But in that package it's the first step in a pipeline for quantitatively comparing color distributions, and provides more options than k means clustering or pre-specified color centers. In a similar vein, images processed in micaToolbox currently require and ImageJ region-of-interest (ROI) mask to restrict an analysis to e.g. just a fish and not the background. Patternize, recolorize, and pavo also all can take background-masked images but provide no method for masking the background besides (if I recall) ignoring particular RGB colors.

The color clustering as shown in the examples also appears to be quite low-information. In the first example, the grouper shown is clearly tinted green, presumably because it was photographed at a depth of at least a few meters down in seawater. So what do we learn by taking the average RGB color of the grouper? The third example (in Fig. 3), which focuses specifically on the color segmentation of the fish, describes the fish-body-only segmentation in 4D as "better matching the original image", but what criteria are we using to say that it is a better match? What information do I gain from the lighter colors around the eye, given how uneven the lighting is in this image to start with (looks like it was taken with flash)? The original recolorize paper specifies that users need some biological criteria for the color classes. I support the use of databases like iNaturalist to get more phenotype information, but we should be explicit about what kind of quantitative data this can realistically provide.

All this to say, I would rather have a really good segmentation package that lets me export segmentation results in a variety of formats to dedicated color clustering softwares than have another color clustering function that does a subset of the analysis found in existing methods. I recommend allowing users to export segmentations as 1) a PNG (or TIFF, etc) with certain regions transparent (e.g. set the background and fins to transparent, export a separate image with each fin separately on a transparent background, etc); 2) a binary mask; 3) an ImageJ ROI (or just a set of XY coordinates); 4) at least one widespread annotation format like COCO JSON. This is up to the author, of course, but these are the export formats I see used most commonly in eco/evo problems (of which I see a lot). It could be quite a powerful but easy-to-generate example to run this package on a small dataset (see below), export the results to another package, and analyze the results from that package, vs. compare how long it takes to run without SegColR (e.g. manual annotation).

3. Demonstrate the method on a small sample dataset

Running the package on a very small sample dataset (even 3-5 images) will enable the author the demonstrate the utility of the package for a concrete problem, even a simple one, as well as test out batch processing options, since this method is really mostly useful in cases where users have more images than they would be willing to label manually (usually 2+ images). These can be taken from iNaturalist, but I would also encourage the author to find a biologist in their home department/university (I am sure the EEB department will have many of them) who needs to quickly label and mask a series of images. Not only would this demonstrate utility for an existing biological problem, but it would help the author troubleshoot the ways that a biologist without extensive computational experience actually uses their package, which will improve uptake in the community. Given the number of bugs I found in testing the package even for the examples, I think getting another user to test out the package could be really valuable. In general, I'm not totally sure why this package is pitched as being primarily useful for images from iNaturalist. All kinds of biological images need a quick segmentation solution.

Specific comments

Code

- Allow users to turn off plotting with `grounded_segmentation_cli` to enable batch processing. Right now (from what I tested) I have to click through every image.
- `plot_seg_results`: This currently throws a warning with the default color palette if there are not at least three classes of labels since the minimum number of color classes for an RColorBrewer palette is 3.

Manuscript

Introduction:

- I obviously am advocating to drop the color analysis, so rather than start by talking about the importance of biological color, I would start by emphasizing how image segmentation is a bottleneck to analysing large image databases for phenotype data. Yes, preservation can induce color loss, but wildly inconsistent lighting conditions, flash, camera variation, etc also fail to preserve color information.
- “Furthermore, manual color extraction can be subject to measurement error, e.g. what one observer calls white another may call light green.” I don’t disagree, but as stated, there is nothing inherently more objective about analyzing the pixel colors of an uncalibrated image of an unknown source. Color is hugely dependent on the available light, the surface, and the spectral sensitivities of the camera. Spatial components - e.g. not what the colors are but *where* the colors are, as well as things like geometric morphometrics (fin/limb/head shape, etc) are much more robust to imaging conditions. Again, I would just focus on the importance of segmentation as a necessary first step to many other analyses.

Section 2.2:

- QCPA is cited as an R package, but it’s an ImageJ plugin.

Section 2.3.2:

- “Traditional color extraction methods applied to the entire image” - Kind of a straw man argument; I don’t think anyone would just analyze all the colors in an uncalibrated image and expect it to be informative. The actual alternative here is that someone would have to spend several minutes outlining this fish in ImageJ or similar, which is not so annoying for one fish, but for even a small dataset of 100 images could take several days. That is (in my view) the real problem that this package is solving.

Section 2.3.4:

- “fine-tuning the model on domain-specific datasets will still be necessary in some cases” - Totally agree, and of course even a very comprehensive general segmentation model is going to struggle with biological imaging problems, which tend to be highly diverse and low in sample size. What do you recommend for fine-tuning? Could be brought up in the discussion.

Discussion:

- “Other issues, such as the shadow distorting color, are more difficult to resolve. Using a predefined color palette with colors that are distant in color space may be one way around this” - I find this unconvincing unless the author can show an example of a dataset with diverse lighting conditions where specifying color classes actually resolves the issue. The last example in the recolorize paper more or less directly shows the opposite (that the same specimen imaged under different lighting conditions appears as different colors). I would delete this claim unless the author can show it working.