img2cmap

Release 0.2.3

Kevin Arvai

Nov 27, 2022

CONTENTS

CHAPTER

ONE

IMG2CMAP

1.1 Usage

Create colormaps from images in three lines of code!

First, `ImageConverter` class converts images to arrays of RGB values.
Then, `generate_cmap` creates a matplotlib ListedColormap.

```
from img2cmap import ImageConverter

# Can be a local file or URL
converter = ImageConverter("tests/images/south_beach_sunset.jpg")
cmap = converter.generate_cmap(n_colors=5, palette_name="south_beach_sunset", random_
↪state=42)
```

Now, use the colormap in your plots!

```
import matplotlib.pyplot as plt

colors = cmap.colors

with plt.style.context("dark_background"):
    for i, color in enumerate(colors):
        plt.plot(range(10), [_+i+1 for _ in range(10)], color=color, linewidth=4)
```

Plot the image and a colorbar side by side.

```
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable

fig, ax = plt.subplots(figsize=(7, 5))
```

(continues on next page)

(continued from previous page)

```
ax.axis("off")
img = plt.imread("tests/images/south_beach_sunset.jpg")
im = ax.imshow(img, cmap=cmap)

divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="10%", pad=0.05)

cb = fig.colorbar(im, cax=cax, orientation="vertical", label=cmap.name)
cb.set_ticks([])
```

1.1.1 Advanced

generate_optimal_cmap

You can extract the optimal number of colors from the image using the `generate_optimal_cmap` method. Under the hood this performs the *elbow method <https://en.wikipedia.org/wiki/Elbow_method_(clustering)>* to determine the optimal number of clusters based on the sum of the squared distances between each pixel and it's cluster center.

```
cmaps, best_n_colors, ssd = converter.generate_optimal_cmap(max_colors=10, random_
↪state=42)

best_cmap = cmaps[best_n_colors]
```

remove_transparent

In an image of the Los Angeles Lakers logo, the background is transparent. These pixels contribute to noise when generating the colors. Running the `remove_transparent` method will remove transparent pixels. Here's a comparison of the colormaps generated by the same image, without and with transparency removed.

Make two ImageConverter objects:

```
from img2cmap import ImageConverter

image_url = "https://loodibee.com/wp-content/uploads/nba-los-angeles-lakers-logo.png"

# Create two ImageConverters, one with transparency removed and one without
converter_with_transparent = ImageConverter(image_url)
converter_with_transparent.remove_transparent()

converter_no_transparent = ImageConverter(image_url)

cmap_with_transparent = converter_with_transparent.generate_cmap(
    n_colors=3, palette_name="with_transparent", random_state=42
```

(continues on next page)

(continued from previous page)

```
)
cmap_no_transparent = converter_no_transparent.generate_cmap(
    n_colors=3, palette_name="no_transparent", random_state=42
)
```

Plot both colormaps with the image:

```
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable

for cmap in [cmap_with_transparent, cmap_no_transparent]:
    fig, ax = plt.subplots(figsize=(7, 5))

    ax.axis("off")
    img = converter_no_transparent.image
    im = ax.imshow(img, cmap=cmap)

    divider = make_axes_locatable(ax)
    cax = divider.append_axes("right", size="10%", pad=0.05)

    cb = fig.colorbar(im, cax=cax, orientation="vertical", label=cmap.name)
    cb.set_ticks([])
```

Notice, only after removing the transparent pixels, does the classic purple and gold show in the colormap.

resize

There is a method of the ImageConverter class to resize images. It will preserve the aspect ratio, but reduce the size of the image.

```
def test_resize():
    imageconverter = ImageConverter("tests/images/south_beach_sunset.jpg")
    imageconverter.resize(size=(512, 512))
    # preserves aspect ratio
    assert imageconverter.image.size == (512, 361)
```

hexcodes

When running the `generate_cmap` or the `generate_optimal_cmap` methods the ImageConverter object will automatically capture the resulting hexcodes from the colormap and store them as an attribute.

```
from img2cmap import ImageConverter

image_url = "https://static1.bigstockphoto.com/3/2/3/large1500/323952496.jpg"

converter = ImageConverter(image_url)
converter.generate_cmap(n_colors=4, palette_name="with_transparent", random_state=42)
print(converter.hexcodes)
```

Output:

```
['#ba7469', '#dfd67d', '#5d536a', '#321e28']
```

1.2 Installation

```
pip install img2cmap
```

You can also install the in-development version with:

```
pip install https://github.com/arvkevi/img2cmap/archive/main.zip
```

1.3 Documentation

https://img2cmap.readthedocs.io/

1.4 Web App

Check out the web app at https://img2cmap.fly.dev

1.5 Status

docs	
tests	
package	

1.6 Development

Install the development requirements:

```
pip install img2cmap[dev]
```

To run all the tests run:

```
tox
```

Note, to combine the coverage data from all the tox environments run:

Windows	`set PYTEST_ADDOPTS=--cov-append` `tox`
Other	`PYTEST_ADDOPTS=--cov-append tox`

CHAPTER

TWO

INSTALLATION

At the command line:

```
pip install img2cmap
```

CHAPTER

THREE

USAGE

To use img2cmap in a project:

```
import img2cmap
```

CHAPTER

FOUR

REFERENCE

4.1 img2cmap

CHAPTER

FIVE

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

5.1 Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.2 Documentation improvements

img2cmap could always use more documentation, whether as part of the official img2cmap docs, in docstrings, or even on the web in blog posts, articles, and such.

5.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/arvkevi/img2cmap/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

5.4 Development

To set up *img2cmap* for local development:

1. Fork img2cmap (look for the "Fork" button).
2. Clone your fork locally:

```
git clone git@github.com:YOURGITHUBNAME/img2cmap.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. Install development requirements:

```
pip install img2cmap[dev]
```

5. When you're done making changes run all the checks and docs builder with tox one command:

```
tox
```

6. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.4.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`).
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

5.4.2 Tips

To run a subset of tests:

```
tox -e envname -- pytest -k test_myfeature
```

To run all the test environments in *parallel*:

```
tox -p auto
```

CHAPTER

SIX

AUTHORS

- Kevin Arvai - https://github.com/arvkevi
- Marshall Krassenstein - https://github.com/mpkrass7

CHAPTER

SEVEN

CHANGELOG

7.1 0.0.0 (2022-04-30)

- First release on PyPI.

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

i

INDEX

I

M