ColorDetect

Release 1.6.0

Marvin Kweyu

Sep 22, 2022

CONTENTS:

CHAPTER

ONE

WELCOME TO COLORDETECT'S DOCUMENTATION

This site covers ColorDetect's usage and module documentation.

CHAPTER

TWO

GETTING STARTED

2.1 Installation

```
$ pip install ColorDetect
```

For usage , import as:

```
import colordetect
```

For more details, proceed to *image_color_recognition*

CHAPTER

THREE

COLORDETECT

3.1 Image color recognition

3.1.1 Example 1

As a walk through some of the capabilities of ColorDetect we will use this sample image.

```
# Get the most dominant color count from an image
>>> from colordetect import ColorDetect
>>>
>>> my_image = ColorDetect("<image_path>")
>>> my_image.get_color_count(color_format="rgb")
'[2.0, 2.0, 249.0]': 6.2, '[5.0, 211.0, 212.0]': 7.15, '[173.0, 25.0, 98.0]': 17.49,
↪'[146.0, 155.0, 9.0]': 18.62, '[253.0, 253.0, 253.0]': 50.54}
```

A dictionary, with the RGB value of the color as the key and its percentage occurrence in the image as the value is returned. To get a more human readable format, one would call `get_color_count()` parsing the parameter for `color_format` as **human_readable**.

Our line to obtain colors would be replaced by:

```
>>> my_image.get_color_count()
{'blue': 6.2, 'darkturquoise': 7.15, 'mediumvioletred': 17.49, 'olive': 18.62, 'white':␣
↪50.54}
```

Note: As of the ColorDetect 0.1.7, the percentage changed from being presented as a key to being presented as a value. This attributed to the uniqueness of python dictionary keys. See the *change log* for more info.

For clarification:

```
'[2.0, 2.0, 249.0]': 6.2
# this key value pair would imply 6.2 % of the image, has an RGB of [2.0, 2.0, 249.0]
```

By default, ColorDetect will count the 5 most dominant colors. This can , of course ,be overridden by parsing an argument specifying how many colors most dominant you need from the image, with values decreasing in their percentage presence the higher you go on the color count.

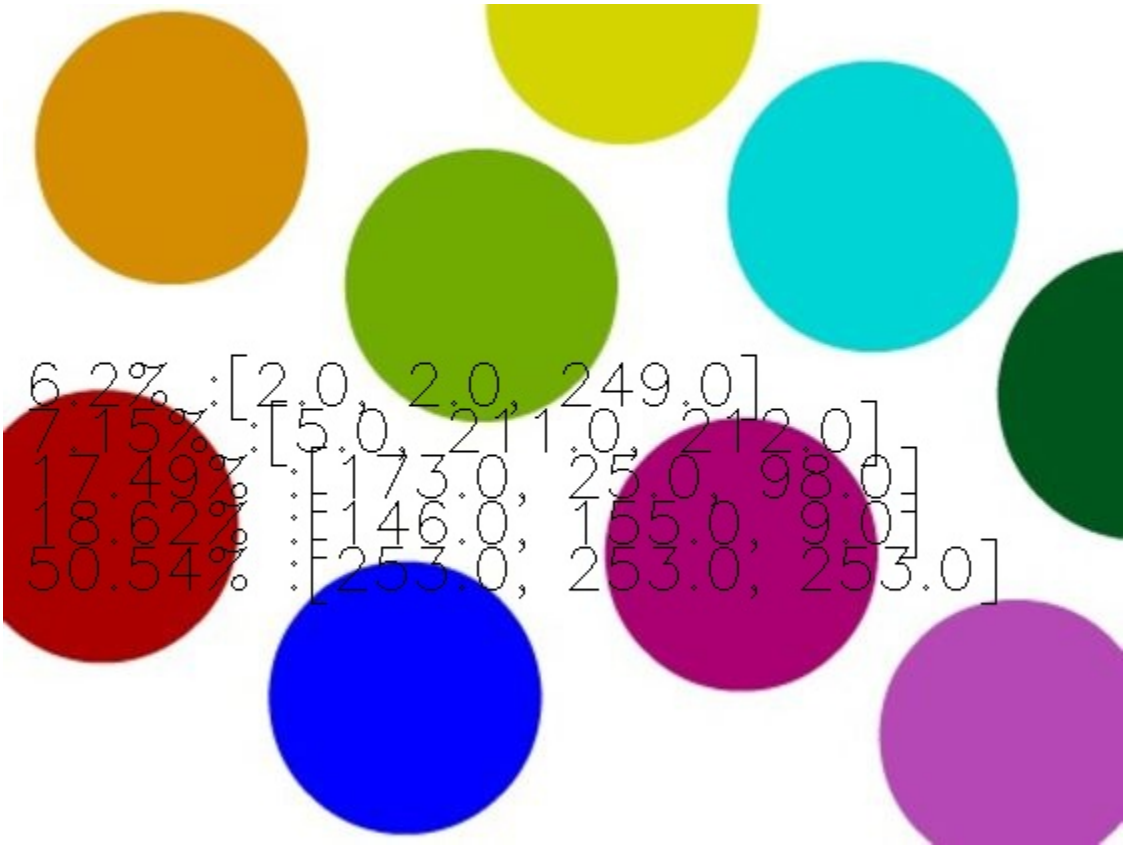
Look up *get_color_count* for details on the different arguments it accepts including the different color format return values. Now suppose you want to take it a step further and write the result to the image itself.

Warning: Take note of the difference in saving the image to storage from the previous *save_color_count<save_color_count>* to *save_image<save_image>*

```
>>> my_image.write_color_count()
>>> my_image.save_image("<path_to_save_image>", "<name_of_image>")
```

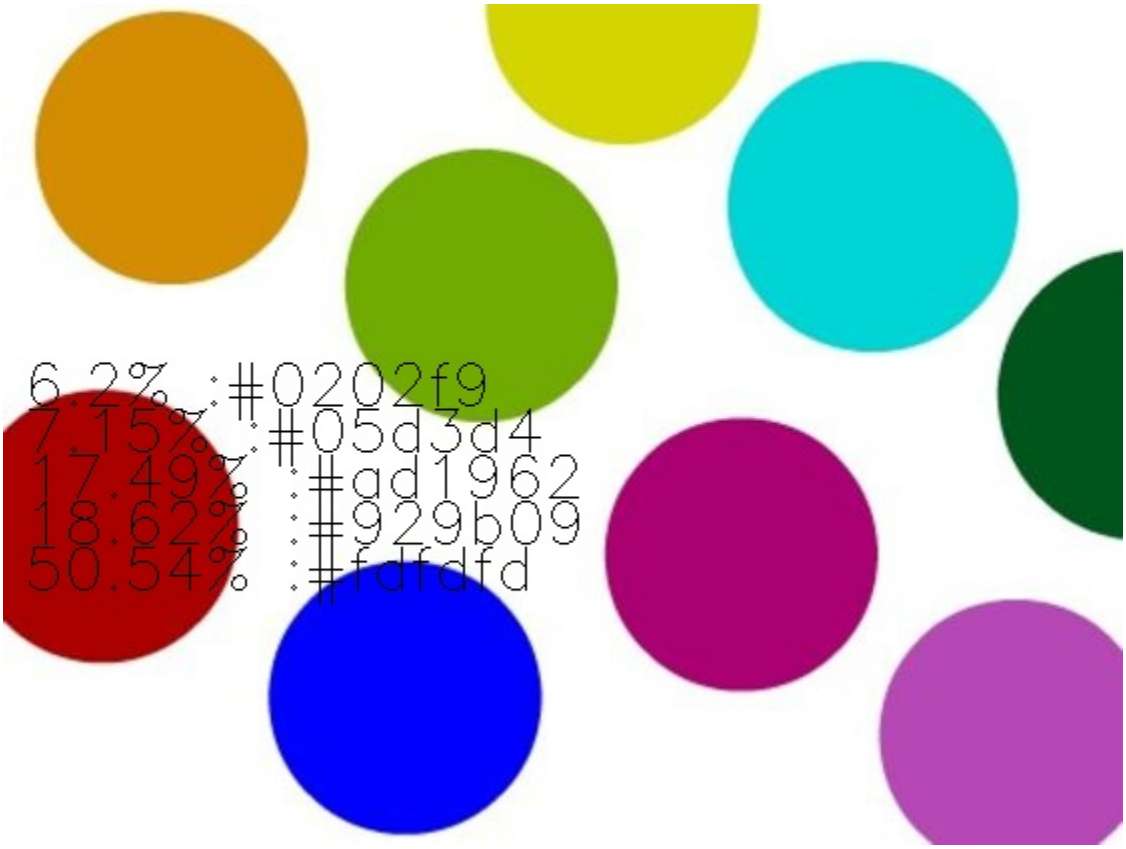
Just as *save_color_count*, save_image will accept , as optional parameters, the path and name of the image with color count on it. By default, these values are `.` (For the current directory the script is being run from) and `out.jpg` respectively.

The result.



Depending on the size of the image, you might want to decide whether to write the count to the image or not. As observed, a smaller image gives a crowded appearance.

As a similar example, with colors represented in their hex format,



Additionally, to enable the use of custom text on an image:

```
>>> from colordetect import ColorDetect
>>> my_image = ColorDetect("<image_path>")
>>> my_image.write_text(text="a random string", font_color=(0,0,0))
```

To appropriately place the text onto the image and ensure the text does not fade over the object on the image with the same color, a font color can be parsed as an RGB tuple. This defaults to *(0,0,0)* , which would be black. More customization features over the text, including text margin, font thickness and line spacing (the space between lines of text) can be found on the *write_text* method documentation.

Whether using `write_text` or `write_color_count`, the image has to be saved using *save_image*.

3.1.2 Getting colors from URL:

```
>>> from colordetect import ColorDetect
>>>
>>> my_image = ColorDetect("<image_url>")
>>> my_image.get_color_count()
```

3.1.3 Example 2

We get colors from a random image on unsplash.

Our photo of choice, is one by Ruby Cevallos on Unsplash

```
>>> from colordetect import ColorDetect
>>>
>>> my_image = ColorDetect("https://images.unsplash.com/photo-1628127437106-0cc010a5fd2d?
↪ixid=MnwxMjA3fDB8MHxlZGl0b3JpYWwtZmVlZHwzfHx8ZW58MHx8fHw%3D&ixlib=rb-1.2.1&auto=format&
↪fit=crop&w=500&q=60")
>>> my_image.get_color_count()
{'saddlebrown': 6.17, 'sienna': 12.62, 'rosybrown': 15.62, 'lightgray': 27.67,
↪'whitesmoke': 37.91}

We may, go ahead and write this color count to the image, and save it.
```

Video color recognition can be done using *VideoColor*

3.2 Color segmentation

3.2.1 Overview

Color segmentation is the process by which specific target parts of the image are extracted based on , in this case , their color.

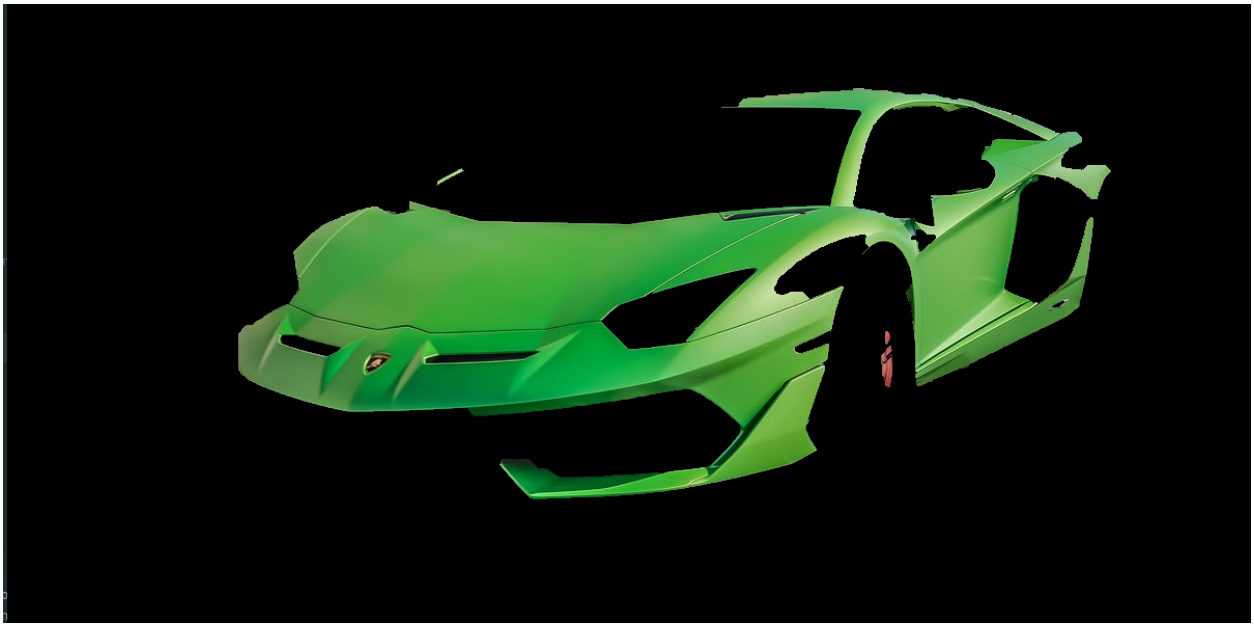
3.2.2 Example

Using the below image, extract the vehicle color. We will display both a segmented image and a monochromatic image(which will exclude the vehicle from the monochrome color scale).

For this, we import *ColorDetect* as below:

```
>>> import cv2
>>> from colordetect import ColorDetect
>>> my_car = ColorDetect('car.jpg')
>>> monochromatic, gray, segmented, mask = my_car.get_segmented_image(lower_bound=(0, 70,
↪ 0), upper_bound=(80, 255, 255))
>>> cv2.imshow('Segmented', segmented)
>>> cv2.imshow('monochromatic', monochromatic)
>>> cv2.wait(0)
```

The lower and upper bounds act as a range of colors from which to look from. as a result, our segmented image would appear as below:



`get_segmented_image()` accepts more parameters such as *erode_iterations,dilate_iterations, use_grab_cut*, which is set True by default and *gc_iterations* . You may increase or decrease these values depending on the clarity needed off the image.

Our monochromatic image:

3.3 Video color recognition

3.3.1 Full video color detection

To show how video color recognition works, the following video of planet earth will be used.

```
>>> from colordetect import VideoColor
>>> my_video = VideoColor("<video_path>")
>>> my_video.get_video_frames()
{'[137.0, 165.0, 182.0]': 0.92, '[71.0, 84.0, 95.0]': 2.16, '[24.0, 30.0, 50.0]': 11.17,
↪'[7.0, 10.0, 26.0]': 17.72, '[0.0, 0.0, 0.0]': 68.83, '[143.0, 170.0, 186.0]': 0.85,
↪'[76.0, 89.0, 100.0]': 2.11, '[26.0, 32.0, 52.0]': 11.07, '[8.0, 11.0, 27.0]': 15.71,
↪'[135.0, 163.0, 181.0]': 0.95, '[76.0, 88.0, 98.0]': 2.05, '[127.0, 160.0, 180.0]': 0.
↪94, '[71.0, 83.0, 95.0]': 2.38, '[7.0, 11.0, 27.0]': 15.72, '[124.0, 159.0, 181.0]': 0.
↪9, '[69.0, 83.0, 95.0]': 2.28, '[26.0, 32.0, 53.0]': 13.73, '[125.0, 160.0, 182.0]': 0.
↪89, '[68.0, 82.0, 95.0]': 2.27, '[132.0, 166.0, 187.0]': 0.79, '[71.0, 87.0, 100.0]':␣
↪2.1, '[25.0, 32.0, 52.0]': 14.18, '[134.0, 167.0, 186.0]': 0.83, '[72.0, 87.0, 100.0]
↪': 2.01, '[26.0, 33.0, 53.0]': 12.11, '[132.0, 165.0, 183.0]': 0.9, '[73.0, 88.0, 99.0]
↪': 2.04, '[8.0, 10.0, 27.0]': 16.76, '[134.0, 166.0, 184.0]': 0.87, '[132.0, 165.0,␣
↪185.0]': 0.86, '[74.0, 89.0, 100.0]': 2.0, '[26.0, 33.0, 52.0]': 10.65, '[7.0, 10.0,␣
↪27.0]': 16.93, '[124.0, 157.0, 178.0]': 0.99, '[68.0, 81.0, 93.0]': 2.14, '[25.0, 31.0,
↪ 50.0]': 10.66, '[124.0, 160.0, 182.0]': 0.88, '[67.0, 82.0, 94.0]': 2.19, '[25.0, 31.
↪0, 49.0]': 10.68, '[124.0, 160.0, 183.0]': 0.85, '[67.0, 83.0, 95.0]': 2.0, '[25.0, 30.
↪0, 49.0]': 11.04, '[123.0, 160.0, 182.0]': 0.87, '[24.0, 29.0, 47.0]': 9.51, '[23.0,␣
↪29.0, 47.0]': 10.6, '[6.0, 9.0, 26.0]': 19.11, '[67.0, 83.0, 97.0]': 2.0, '[24.0, 29.0,
```

(continues on next page)

(continued from previous page)

```
↪ 48.0]': 9.83, '[125.0, 161.0, 183.0]': 0.88, '[67.0, 83.0, 96.0]': 1.96, '[127.0, 162.
↪0, 183.0]': 0.87, '[23.0, 29.0, 46.0]': 8.58, '[5.0, 8.0, 25.0]': 17.77, '[68.0, 84.0,␣
↪98.0]': 1.9, '[24.0, 29.0, 46.0]': 6.95, '[125.0, 161.0, 184.0]': 0.85, '[67.0, 84.0,␣
↪99.0]': 1.89, '[133.0, 165.0, 186.0]': 0.82, '[67.0, 85.0, 99.0]': 1.84, '[23.0, 28.0,␣
↪45.0]': 6.83, '[5.0, 8.0, 24.0]': 22.22, '[135.0, 165.0, 186.0]': 0.85, '[69.0, 86.0,␣
↪100.0]': 1.79, '[22.0, 27.0, 43.0]': 7.22, '[5.0, 7.0, 24.0]': 22.48, '[133.0, 166.0,␣
↪186.0]': 0.81, '[73.0, 91.0, 105.0]': 1.69, '[129.0, 163.0, 185.0]': 0.85, '[69.0, 86.
↪0, 98.0]': 1.9, '[21.0, 27.0, 44.0]': 7.25, '[4.0, 7.0, 24.0]': 21.7, '[68.0, 86.0,␣
↪101.0]': 1.9, '[22.0, 27.0, 45.0]': 7.91, '[126.0, 160.0, 181.0]': 0.94, '[66.0, 83.0,␣
↪96.0]': 1.91, '[22.0, 27.0, 46.0]': 9.19, '[129.0, 163.0, 184.0]': 0.86, '[68.0, 85.0,␣
↪98.0]': 2.01, '[21.0, 27.0, 46.0]': 10.62, '[133.0, 165.0, 185.0]': 0.85, '[69.0, 86.0,
↪ 99.0]': 1.96, '[23.0, 29.0, 48.0]': 10.61, '[7.0, 9.0, 26.0]': 17.7, '[135.0, 165.0,␣
↪185.0]': 0.85, '[73.0, 88.0, 100.0]': 1.96, '[24.0, 29.0, 50.0]': 11.34, '[139.0, 164.
↪0, 177.0]': 0.92}
```

Just as image color recognition, a dictionary will be returned. *get_video_frames* takes optional parameters, that is, *frame_color_count*, an integer describing how many colors to get per frame grabbed, and *color_format*, working much the same way as *get_color_count*, which is to say either RGB, HSV or hex values.

Depending on the video, you may want to display progress of the processing. Thus, an additional, optional parameter, `progress=True` may be included. This is **False** by default.

Colors are grabbed on a per second basis. Hence , in a video 30 seconds long, a single frame will be used for each second of feed.

Have a look at *col_share* for details into how you may format the results.

3.3.2 Working with videos and time

We can get colors at specific times of the parsed video

```
>>> from colordetect import VideoColor
>>> my_video = VideoColor("<video_path>")
>>> (image, color_description) = my_video.get_time_frame_color(time=15000)
```

The result is a tuple with a ColorDetect object and a color description. We can proceed to save the color description onto the image in our preferred color

```
>>> image.write_color_count(font_color=(255,255,255), save=True)
```

Locate a file *out.jpg* in your current working directory.

We could, **alternatively**, handle the saving ourselves and go as below:

```
>>> image.write_color_count(font_color=(255,255,255))
>>> image.save_image(location='path/to/directory/of/choice', filename='filenameofchoice.
↪jpg')
```

3.4 col_share

Depending on how many colors you have or how many dominant colors you want to narrow down to:

```
>>> from colordetect import col_share
>>> all_colors = my_video.get_video_frames()
>>> top_colors = col_share.sort_order(object_description=all_colors,key_count=5)
{'[0.0, 0.0, 0.0]': 68.83, '[5.0, 7.0, 24.0]': 22.48, '[5.0, 8.0, 24.0]': 22.22, '[4.0,␣
↪7.0, 24.0]': 21.7, '[6.0, 9.0, 26.0]': 19.11}
```

The sort gets the top 5 colors, by default, from all the colors obtained from all the frames present. This may be adjusted to suit your needs. A reverse of the same may be obtained by passing the *ascending* parameter and setting this to false: `ascending=False`.

```
>>> top_colors = col_share.sort_order(object_description=all_colors,key_count=5,␣
↪ascending=False)
```

CHAPTER

FOUR

MISCELLANEOUS

Helpful pages.

4.1 Developer Notes

Here, we dive into ColorDetect's inner definitions and working. Found a bug or feature request you would like to address? Take a look at the *Contribution guidelines* and feel free to submit a pull. The project source is hosted on Github

4.1.1 Module ColorDetect

Defines ColorDetect class

For example:

```
>>> from colordetect import ColorDetect
>>> user_image = ColorDetect("<path_to_image>")
# where color_count is the target most dominant colors to be found. Default set to 5
>>> colors =  user_image.get_color_count(color_count=5)
>>> colors
# alternatively, save these RGB values to the image
>>> user_image.write_color_count()
>>> user_image.save_image("<storage_path>","<image_file_name>")
# Image processed and saved successfully
```

class colordetect.color_detect.**ColorDetect**(*image*, *resize_h: Optional[int] = None*)

Bases: `object`

Detect and recognize the number of colors in an image

get_color_count(*color_count: int = 5*, *color_format: str = 'human_readable'*) → dict

Count the number of different colors

color_count: int
The number of most dominant colors to be obtained from the image

color_format:str
The format to return the color in. Options

- hsv - (60°,100%,100%)
- rgb - rgb(255, 255, 0) for yellow

- hex - #FFFF00 for yellow
- human_readable - yellow for yellow

Returns
color description

`get_segmented_image`(*lower_bound: tuple*, *upper_bound: tuple*, *erode_iterations: int = 3*, *dilate_iterations: int = 3*, *use_grab_cut: bool = True*, *gc_iterations: int = 3*) → tuple

Get image masks from an image

lower_bound: tuple
A lower color range from which to look from

upper_bound: tuple
The higher RGB color range from which to look from

erode_iterations: int
The number of times to perform erosion of the image

dilate_iterations: int
The number of times dilation is applied.

use_grab_cut: bool
A boolean indicating whether grabCut will be applied to the image. This is True by default.

gc_iterations: int
Number of iterations the algorithm should make before returning the result

Returns
output_image, gray, segmented, mask

`save_image`(*location: str = '.'*, *file_name: str = 'out.jpg'*)

Save the resultant image file to the local directory

location: str
The file location of the image

file_name:str
The name of the new image

`write_color_count`(*left_margin: int = 10*, *top_margin: int = 20*, *font: int = 0*, *font_color: tuple = (0, 0, 0)*, *font_scale: float = 1.0*, *font_thickness: float = 1*, *line_type: int = 1*, *save: bool = False*)

Write the number of colors found to the image

left_margin: int
Text spacing from the left

top_margin: int
Text spacing from the top

font: int
Font to use in text. Look up acceptable values from python-opencv

font_color:
RGB tuple of text font color

font_scale:
Size of the text to be written

font_thickness:
Thickness of the text

line_type: int = 1,

write_text(*text: str = ''*, *left_margin: int = 10*, *top_margin: int = 20*, *font: int = 0*, *font_color: tuple = (0, 0, 0)*, *font_scale: float = 1.0*, *font_thickness: float = 1.0*, *line_type: int = 1*, *line_spacing: int = 0*)

Write text onto an image

Parameters

text: str
The text to be written onto the image

line_spacing:int
The spacing between lines

left_margin: int
Text spacing from the left

top_margin: int
Text spacing from the top

font: int
Font to use in text. Look up acceptable values from python-opencv

font_color:
RGB tuple of text font color

font_scale:
Size of the text to be written

font_thickness: float = 1.0
Thickness of the text

line_type: int = 1,
Space betweeen the lines

Returns

4.1.2 Module VideoColor

Defines VideoColor class

Usage:

```
>>> from colordetect import VideoColor
>>> user_video = VideoColor("<path_to_video>")
# where frame_color_count is the target most dominant colors to be found. Default set to␣
↪5
>>> colors =  user_video.get_video_frames(frame_color_count=7)
>>> colors
# alternatively shorten the dictionary to get a specific number of sorted colors from␣
↪the whole lot
>>> from colordetect import col_share
>>> top_colors = col_share.sort_order(object_description=colors, key_count=8)
```

class colordetect.video_color_detect.**VideoColor**(*video*)

Bases: object

Detect and recognize the number of colors in a video

get_time_frame_color(*color_count: int = 5*, *color_format: str = 'rgb'*, *time: int = 1000*) → tuple

Get color from a specific time in the video

Parameters

time: int
Time to get color from in video in milliseconds

color_count: int
Number of colors to return at the given time frame

color_format:str

The format to return the color in. Options

- hsv - (60°,100%,100%)
- rgb - rgb(255, 255, 0) for yellow
- hex - #FFFF00 for yellow

return
(image, color_description)

get_video_frames(*frame_color_count: int = 5*, *color_format: str = 'rgb'*, *progress: bool = False*) → dict

Get image frames and their colors from the video

frame_color_count: int
The number of most dominant colors to be obtained from a single frame

color_format:str

The format to return the color in. Options

- hsv - (60°,100%,100%)
- rgb - rgb(255, 255, 0) for yellow
- hex - #FFFF00 for yellow

return
color_description dictionary

4.1.3 Module ColShare

Global methods non-object specific

Usage:

```
>>> from colordetect import col_share
# show a progress bar for a process
>>> col_share.progress_bar("<current_process_position>", "<total_process_length>", "
↪<process_description>")
# sort a dictionary by value to required length or in specific order
>>> col_share.sort_order('<dictionary>', "<items_to_return>", "<order>")
```

colordetect.col_share.**is_url**(*url: str*) → bool

Check if the string parsed is a URL

url: str
A string to be checked

colordetect.col_share.**progress_bar**(*position: int*, *total_length: int*, *post_text: str = 'Color Detection'*)

Display a progress bar of video processing

position: int
Current position of process

total_length: int
Total length of process

post_text: str
Text to display along with progress bar

colordetect.col_share.**sort_order**(*object_description: dict*, *key_count: int = 5*, *ascending: bool = True*)

Sort items in a dictionary according to value

object_description: dict
A dictionary whose values need sorting

key_count: int
The number of items to return from the sort

ascending: bool
The order to perform the dictionary sort. By default, set to True.

Returns
A sorted dictionary with specific number

4.2 Contributing to ColorDetect

Thank you for taking your time to look at the ColorDetect project.

Use the following as guidelines to making your contributions and do feel free to propose changes to this document in a pull request. The source code, located at the ColorDetect project page.

4.2.1 Setup

This project can be setup with:

```
python3 -m venv .venv
.venv/bin/activate
pip install -r requirements/requirements-dev.txt
pre-commit install
```

4.2.2 Issues

Check if the issue has been addressed or is in progress and if not , only then do you create a new issue. Remember to give it the appropriate label

4.2.3 Enhancements

Describe the enhancement in mind and what you would expect to have resulted from this process. Submit the enhancement with the *enhancement* tag **along with its test**

4.2.4 Pull requests

Reference the issue or enhancement being referenced in the pull request and submit the pull request to the *development* branch.

4.3 ColorDetect Changelog

4.3.1 1.6.0 (22-09-2022)

4.3.1.1 Features

- Perform color recognition on a video at a specific time
- Extract image from video at a specific time

4.3.2 1.5.1 (30-08-2021)

4.3.2.1 Docs

- Update contribution readme with pre-commit configuration.

4.3.2.2 Fix

- Linting of code

4.3.3 1.5.0 (11-08-2021)

4.3.3.1 Features

- Perform color recognition on images based on URL passed

4.3.4 1.4.6 (07-09-2021)

4.3.4.1 Docs

- Contribution pre-commit file linting

4.3.4.2 Bug Fixes

- Tests passing locally and failing remotely

4.3.5 1.4.5 (06-28-2021)

4.3.5.1 Bug Fixes

- Documentation update

4.3.6 1.4.3 (06-28-2021)

4.3.6.1 Bug Fixes

- Video processing percentage display fix

4.3.7 1.4.3 (03-29-2021)

4.3.7.1 Bug Fixes

- Validate RGB font color input and add tests for it

4.3.8 1.4.2 (03-09-2021)

4.3.8.1 Bug Fixes

- Input RGB values instead of inverted BGR in writing color count and text

4.3.9 1.4.1 (02-09-2021)

4.3.9.1 Documentation

- Fix error in image display of masked image
- Format documentation to fix side panel and have structure in sections

4.3.10 1.4.0 (02-09-2021)

4.3.10.1 Features

- Image color segmentation, masking and monochromatic colors on specific image sections

4.3.11 1.3.0 (02-02-2021)

4.3.11.1 Features

- Add a return of human readable colors.

4.3.11.2 Documentation

- Update ColorDetect module documentation to show method params

4.3.12 1.3.0rc (18-01-2021)

4.3.12.1 Features

- Add a return of human readable colors.

4.3.12.2 Documentation

- Update ColorDetect module documentation to show method params
- Move to version `1.3.0rc` due to error in `1.1.1` packaging

4.3.13 1.1.1 (17-01-2021)

4.3.13.1 Documentation

- Update setup to show correct package version.

4.3.14 1.1.0 (17-01-2021)

4.3.14.1 Features

- Enable customization of text input from the user as well as color count being written to the image

4.3.14.2 Documentation

- Add contributors to readme and update project documentation with relevant parameter methods

4.3.15 1.0.1 (23-11-2020)

4.3.15.1 Features

- Add pre-commit hooks for better contribution styling

4.3.15.2 Documentation

- Update readme with development guide.

4.3.16 1.0.0 (03-10-2020)

4.3.16.1 Features

- Creation of col_share module. Split methods non-exclusive to *VideoColor* and *ColorDetect*

4.3.16.2 Documentation

- Include col_share documentation.
- Update readme to reflect col_share.

4.3.17 0.3.1 (17-10-2020)

4.3.17.1 Bug fix

- Perform check to ensure the color description has content before writing color count.

4.3.18 0.3.0 (26-09-2020)

4.3.18.1 Features

- Video color detection and recognition

4.3.18.2 Documentation

- Include video color detection documentation
- Correction in package imports

4.3.19 0.2.0 (13-08-2020)

4.3.19.1 Features

- Enable input of custom text onto the image

4.3.19.2 Documentation

- Add `write_text` method along with other breaking changes to the documentation

4.3.20 0.1.7 (17-04-2020)

4.3.20.1 Features

- Invert return of recognized colors dictionary. Return the colors as keys and percentages as values to avoid duplicate dictionary keys.

4.3.21 0.1.6 (17-04-2020)

4.3.21.1 Features

- Add color format return options. Include RGB, hex and hsv

4.3.21.2 Misc

- Add tests suite and move test files out of project root.
- Add contributions file
- Update dev requirements
- Improve methods types specification and exception catching.

4.3.22 0.1.5 (11-04-2020)

4.3.22.1 Features

- Return a whole number for the RGB value instead of float.

4.3.22.2 Documentation

- Add changelog to the documentation.

4.3.23 0.1.4 (5-04-2020)

4.3.23.1 Features

- Allow recognition of non pre-defined color sets
- Allow a plain dictionary to be obtained with color recognition from the image before writing onto it.
- Format display of percentage and RGB values

4.3.23.2 Bugfixes

- Update CI config file with correct requirements path.
- Correct test running instructions on README.

4.3.23.3 Improved Documentation

- Publish package documentation ColorDetect

4.3.23.4 Misc

- Add versioning to readme and edit dev requirements.

4.3.24 0.1.3 (22-03-2020)

4.3.24.1 Features

- Change image reading from command-line to ColorDetect object initialization.

4.3.24.2 Bug fixes

- Fix image reading.

4.3.24.3 Misc

- Split dev and base requirements.

4.3.25 0.1.2 (22-03-2020)

4.3.25.1 Features

- Include project license

4.3.26 0.1.1 (22-03-2020)

- Initial release

4.4 Indices and tables

- genindex
- modindex
- search

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX