Colour - Checker Detection Documentation

Release 0.2.0

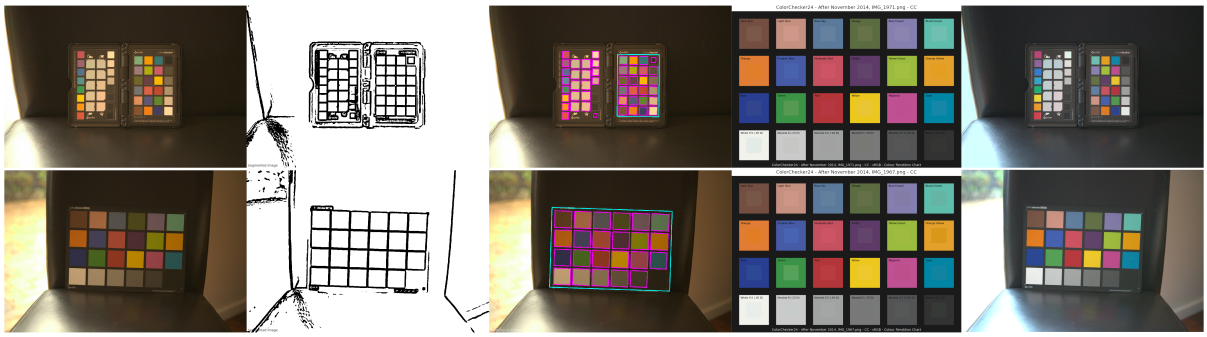
Colour Developers

Jan 11, 2024

CONTENTS

A Python package implementing various colour checker detection algorithms and related utilities.

It is open source and freely available under the BSD-3-Clause terms.



CHAPTER

ONE

1.1 FEATURES

The following colour checker detection algorithms are implemented:

- Segmentation
- Machine learning inference via Ultralytics YOLOv8
  - The model is published on HuggingFace, and was trained on a purposely constructed dataset.
  - The model has only been trained on *ColorChecker Classic 24* images and will not work with *ColorChecker Nano* or *ColorChecker SG* images.
  - Inference is performed by a script licensed under the terms of the *GNU Affero General Public License v3.0* as it uses the *Ultralytics YOLOv8* API which is incompatible with the *BSD-3-Clause*.

1.1 1.1.1 Examples

Various usage examples are available from the examples directory.

CHAPTER

TWO

1.2 USER GUIDE

2.1 User Guide

The user guide provides an overview of **Colour - Checker Detection** and explains important concepts and features, details can be found in the API Reference.

2.1.1 Installation Guide

Because of their size, the resources dependencies needed to run the various examples and unit tests are not provided within the Pypi package. They are separately available as Git Submodules when cloning the repository.

Primary Dependencies

Colour - Checker Detection requires various dependencies in order to run:

- python >= 3.8, < 4
- colour-science >= 4.3
- imageio >= 2, < 3
- numpy >= 1.22, < 2
- opencv-python >= 4, < 5
- scipy >= 1.8, < 2

Secondary Dependencies

- ultralytics >= 8, < 9

Pypi

Once the dependencies are satisfied, **Colour - Checker Detection** can be installed from the Python Package Index by issuing this command in a shell:

```
pip install --user colour-checker-detection
```

The tests suite dependencies are installed as follows:

```
pip install --user 'colour-checker-detection[tests]'
```

The documentation building dependencies are installed as follows:

```
pip install --user 'colour-checker-detection[docs]'
```

The overall development dependencies are installed as follows:

```
pip install --user 'colour-checker-detection[development]'
```

2.1.2 Bibliography

CHAPTER

THREE

1.3 API REFERENCE

3.1 API Reference

3.1.1 Colour - Checker Detection

Colour Checker Detection

Inference

colour_checker_detection

SETTINGS_INFERENCE_COLORCHECKER_CLASSIC	Settings for the inference of the *X-Rite ColorChecker Classic*.
SETTINGS_INFERENCE_COLORCHECKER_CLASSIC_MINI	Settings for the inference of the *X-Rite ColorChecker Classic Mini*.
inferencer_default(image[, cctf_encoding, ...])	Predict the colour checker rectangles in given image using *Ultralytics YOLOv8*.
detect_colour_checkers_inference(image[, ...])	Detect the colour checkers swatches in given image using inference.

colour_checker_detection.SETTINGS_INFERENCE_COLORCHECKER_CLASSIC

colour_checker_detection.**SETTINGS_INFERENCE_COLORCHECKER_CLASSIC = {'aspect_ratio': 1.4285714285714286, 'inferred_class': 'ColorCheckerClassic24', 'inferred_confidence': 0.85, 'interpolation_method': 2, 'reference_values': array([[0.17355167, 0.07874029, 0.05326058], [0.55946176, 0.27734355, 0.21194777], [0.10509124, 0.18955202, 0.32693865], [0.10506442, 0.15021316, 0.05221047], [0.22885963, 0.21350031, 0.42346758], [0.11449231, 0.50663347, 0.41229432], [0.74499115, 0.20172072, 0.0325174], [0.0606182 , 0.10259253, 0.38373146], [0.56055825, 0.08072134, 0.11432307], [0.10983077, 0.04254067, 0.13682661], [0.32967574, 0.49495612, 0.04886544], [0.7689789 , 0.35655545, 0.02534346], [0.0225082 , 0.04870543, 0.28081679], [0.0444356 , 0.29068277, 0.06458335], [0.44636923, 0.03676343, 0.0406788], [0.83803037, 0.57175305, 0.01273052], [0.52392518, 0.07924915, 0.28656418], [-0.04308491, 0.23415773, 0.37506175], [0.87919095, 0.88476747, 0.8349529], [0.58443959, 0.59212352, 0.58458201], [0.35767777, 0.36706043, 0.36528718], [0.19008669, 0.19086038, 0.1898278], [0.08593528, 0.08873843, 0.08978779], [0.03135966, 0.03149993, 0.03231098]]), 'swatches': 24, 'swatches_achromatic_slice': slice(19, 23, 1), 'swatches_chromatic_slice': slice(1, 5, 1), 'swatches_horizontal': 6, 'swatches_vertical': 4, 'transform': {'rotation': 0, 'scale': array([1. , 1.05]), 'translation': array([0, 0])}, 'working_height': 1008, 'working_width': 1440}**

Settings for the inference of the *X-Rite ColorChecker Classic*.

colour_checker_detection.SETTINGS_INFERENCE_COLORCHECKER_CLASSIC_MINI

colour_checker_detection.**SETTINGS_INFERENCE_COLORCHECKER_CLASSIC_MINI = {'aspect_ratio': 1.7094017094017093, 'inferred_class': 'ColorCheckerSG', 'inferred_confidence': 0.85, 'interpolation_method': 2, 'reference_values': array([[0.17355167, 0.07874029, 0.05326058], [0.55946176, 0.27734355, 0.21194777], [0.10509124, 0.18955202, 0.32693865], [0.10506442, 0.15021316, 0.05221047], [0.22885963, 0.21350031, 0.42346758], [0.11449231, 0.50663347, 0.41229432], [0.74499115, 0.20172072, 0.0325174], [0.0606182 , 0.10259253, 0.38373146], [0.56055825, 0.08072134, 0.11432307], [0.10983077, 0.04254067, 0.13682661], [0.32967574, 0.49495612, 0.04886544], [0.7689789 , 0.35655545, 0.02534346], [0.0225082 , 0.04870543, 0.28081679], [0.0444356 , 0.29068277, 0.06458335], [0.44636923, 0.03676343, 0.0406788], [0.83803037, 0.57175305, 0.01273052], [0.52392518, 0.07924915, 0.28656418], [-0.04308491, 0.23415773, 0.37506175], [0.87919095, 0.88476747, 0.8349529], [0.58443959, 0.59212352, 0.58458201], [0.35767777, 0.36706043, 0.36528718], [0.19008669, 0.19086038, 0.1898278], [0.08593528, 0.08873843, 0.08978779], [0.03135966, 0.03149993, 0.03231098]]), 'swatches': 24, 'swatches_achromatic_slice': slice(19, 23, 1), 'swatches_chromatic_slice': slice(1, 5, 1), 'swatches_horizontal': 6, 'swatches_vertical': 4, 'transform': {'rotation': 0, 'scale': array([1.15, 1.]), 'translation': array([0, 0])}, 'working_height': 842, 'working_width': 1440}**

Settings for the inference of the *X-Rite ColorChecker Classic Mini*.

colour_checker_detection.inferencer_default

colour_checker_detection.**inferencer_default**(*image: str | ArrayLike*, *cctf_encoding: Callable = eotf_inverse_sRGB*, *apply_cctf_encoding: bool = True*, *show: bool = False*) → NDArrayInt | NDArrayFloat

Predict the colour checker rectangles in given image using *Ultralytics YOLOv8*.

Parameters

- **image** (str | ArrayLike) – Image (or image path to read the image from) to detect the colour checker rectangles from.
- **cctf_encoding** (Callable) – Encoding colour component transfer function / opto-electronic transfer function used when converting the image from float to 8-bit.
- **apply_cctf_encoding** (bool) – Apply the encoding colour component transfer function / opto-electronic transfer function.
- **show** (bool) – Whether to show various debug images.

Returns

Array of inference results as rows of confidence, class, and mask.

Return type

np.ndarray

Warning: This definition sub-processes to a script licensed under the terms of the *GNU Affero General Public License v3.0* as it uses the *Ultralytics YOLOv8* API which is incompatible with the *BSD-3-Clause*.

Examples

```
>>> import os
>>> from colour import read_image
>>> from colour_checker_detection import ROOT_RESOURCES_TESTS
>>> path = os.path.join(
...     ROOT_RESOURCES_TESTS,
...     "colour_checker_detection",
...     "detection",
...     "IMG_1967.png",
... )
>>> results = inferencer_default(path)
>>> results.shape
(1, 3)
>>> results[0][0]
array(0.9708795...)
>>> results[0][1]
array(0.0...)
>>> results[0][2].shape
(864, 1280)
```

colour_checker_detection.detect_colour_checkers_inference

colour_checker_detection.**detect_colour_checkers_inference**(*image: str | ArrayLike*, *samples: int = 32*, *cctf_decoding=eotf_sRGB*, *apply_cctf_decoding: bool = False*, *inferencer: Callable = inferencer_default*, *inferencer_kwargs: dict | None = None*, *show: bool = False*, *additional_data: bool = False*, ***kwargs: Any*) → Tuple[DataDetectionColourChecker | NDArrayFloat, ...]

Detect the colour checkers swatches in given image using inference.

Parameters

- **image** (str | ArrayLike) – Image (or image path to read the image from) to detect the colour checker rectangles from.
- **samples** (int) – Sample count to use to average (mean) the swatches colours. The effective sample count is $samples^2$.
- **cctf_decoding** – Decoding colour component transfer function / opto-electronic transfer function used when converting the image from 8-bit to float.
- **apply_cctf_decoding** (bool) – Apply the decoding colour component transfer function / opto-electronic transfer function.
- **inferencer** (Callable) – Callable responsible to make predictions on the image and extract the colour checker rectangles.
- **inferencer_kwargs** (dict | None) – Keyword arguments to pass to the inferencer.
- **show** (bool) – Whether to show various debug images.
- **additional_data** (bool) – Whether to output additional data.
- **aspect_ratio** – Colour checker aspect ratio, e.g. 1.5.

- **aspect_ratio_minimum** – Minimum colour checker aspect ratio for detection: projective geometry might reduce the colour checker aspect ratio.
- **aspect_ratio_maximum** – Maximum colour checker aspect ratio for detection: projective geometry might increase the colour checker aspect ratio.
- **swatches** – Colour checker swatches total count.
- **swatches_horizontal** – Colour checker swatches horizontal columns count.
- **swatches_vertical** – Colour checker swatches vertical row count.
- **swatches_count_minimum** – Minimum swatches count to be considered for the detection.
- **swatches_count_maximum** – Maximum swatches count to be considered for the detection.
- **swatches_chromatic_slice** – A *slice* instance defining chromatic swatches used to detect if the colour checker is upside down.
- **swatches_achromatic_slice** – A *slice* instance defining achromatic swatches used to detect if the colour checker is upside down.
- **swatch_minimum_area_factor** – Swatch minimum area factor f with the minimum area m_a expressed as follows: $m_a = image_w * image_h / s_c / f$ where $image_w$, $image_h$ and s_c are respectively the image width, height and the swatches count.
- **swatch_contour_scale** – As the image is filtered, the swatches area will tend to shrink, the generated contours can thus be scaled.
- **working_width** – Size the input image is resized to for detection.
- **fast_non_local_means_denoising_kwargs** – Keyword arguments for `cv2.fastNlMeansDenoising()` definition.
- **adaptive_threshold_kwargs** – Keyword arguments for `cv2.adaptiveThreshold()` definition.
- **interpolation_method** – Interpolation method used when resizing the images, *cv2.INTER_CUBIC* and *cv2.INTER_LINEAR* methods are recommended.
- **kwargs** (`Any`) –

Returns
Tuple of `DataDetectionColourChecker` class instances or colour checkers swatches.

Return type
class`tuple`

Examples

```
>>> import os
>>> from colour import read_image
>>> from colour_checker_detection import ROOT_RESOURCES_TESTS
>>> path = os.path.join(
...     ROOT_RESOURCES_TESTS,
...     "colour_checker_detection",
...     "detection",
...     "IMG_1967.png",
... )
>>> image = read_image(path)
>>> detect_colour_checkers_inference(image)
(array([[ 0.3602327 ,  0.22158547,  0.11813926],
       [ 0.62800723,  0.39357048,  0.24196433],
```

(continues on next page)

(continued from previous page)

```
       [ 0.3284166 ,  0.31669423,  0.28818974],
       [ 0.3072932 ,  0.2744136 ,  0.10451803],
       [ 0.4204691 ,  0.31953654,  0.30901137],
       [ 0.34471545,  0.44057423,  0.29297924],
       [ 0.678418  ,  0.35242617,  0.06670552],
       [ 0.27259055,  0.2535471 ,  0.32912973],
       [ 0.6190633 ,  0.27043283,  0.18543543],
       [ 0.30721852,  0.18180828,  0.19161244],
       [ 0.4858081 ,  0.46007228,  0.03085822],
       [ 0.6499356 ,  0.4018961 ,  0.01579806],
       [ 0.19425018,  0.18621376,  0.27193058],
       [ 0.27500305,  0.38600868,  0.1245231 ],
       [ 0.55459476,  0.21477987,  0.12434786],
       [ 0.71898675,  0.5149239 ,  0.00561224],
       [ 0.5787967 ,  0.25837064,  0.2693373 ],
       [ 0.1743919 ,  0.31709513,  0.29550385],
       [ 0.7383609 ,  0.60645705,  0.43850273],
       [ 0.62609893,  0.5172464 ,  0.36816722],
       [ 0.5117422 ,  0.4191487 ,  0.3013721 ],
       [ 0.36412936,  0.2987345 ,  0.20754097],
       [ 0.26675388,  0.21421173,  0.14176223],
       [ 0.15856811,  0.13483825,  0.07938566]], dtype=float32),)
```

Segmentation

colour_checker_detection

SETTINGS_SEGMENTATION_COLORCHECKER_CLASSIC	Settings for the segmentation of the *X-Rite ColorChecker Classic* and *X-Rite ColorChecker Passport*.
SETTINGS_SEGMENTATION_COLORCHECKER_SG	Settings for the segmentation of the *X-Rite ColorChecker SG**.
SETTINGS_SEGMENTATION_COLORCHECKER_NANO	Settings for the segmentation of the *X-Rite ColorChecker Nano**.
segmenter_default(image[, cctf_encoding, ...])	Detect the colour checker rectangles in given image $image$ using segmentation.
detect_colour_checkers_segmentation(image[, ...])	Detect the colour checkers swatches in given image using segmentation.

colour_checker_detection.SETTINGS_SEGMENTATION_COLORCHECKER_CLASSIC

colour_checker_detection.**SETTINGS_SEGMENTATION_COLORCHECKER_CLASSIC = {'aspect_ratio': 1.5, 'aspect_ratio_maximum': 1.6500000000000001, 'aspect_ratio_minimum': 1.35, 'interpolation_method': 2, 'reference_values': array([[0.17355167, 0.07874029, 0.05326058], [0.55946176, 0.27734355, 0.21194777], [0.10509124, 0.18955202, 0.32693865], [0.10506442, 0.15021316, 0.05221047], [0.22885963, 0.21350031, 0.42346758], [0.11449231, 0.50663347, 0.41229432], [0.74499115, 0.20172072, 0.0325174], [0.0606182 , 0.10259253, 0.38373146], [0.56055825, 0.08072134, 0.11432307], [0.10983077, 0.04254067, 0.13682661], [0.32967574, 0.49495612, 0.04886544], [0.7689789 , 0.35655545, 0.02534346], [0.0225082 , 0.04870543, 0.28081679], [0.0444356 , 0.29068277, 0.06458335], [0.44636923, 0.03676343, 0.0406788], [0.83803037, 0.57175305, 0.01273052], [0.52392518, 0.07924915, 0.28656418], [-0.04308491, 0.23415773, 0.37506175], [0.87919095, 0.88476747, 0.8349529], [0.58443959, 0.59212352, 0.58458201], [0.35767777, 0.36706043, 0.36528718], [0.19008669, 0.19086038, 0.1898278], [0.08593528, 0.08873843, 0.08978779], [0.03135966, 0.03149993, 0.03231098]]), 'swatch_contour_scale': 1.3333333333333333, 'swatch_minimum_area_factor': 200, 'swatches': 24, 'swatches_achromatic_slice': slice(19, 23, 1), 'swatches_chromatic_slice': slice(1, 5, 1), 'swatches_count_maximum': 30, 'swatches_count_minimum': 18, 'swatches_horizontal': 6, 'swatches_vertical': 4, 'transform': {}, 'working_height': 960, 'working_width': 1440}**

Settings for the segmentation of the *X-Rite ColorChecker Classic* and *X-Rite ColorChecker Passport*.

colour_checker_detection.SETTINGS_SEGMENTATION_COLORCHECKER_SG

colour_checker_detection.**SETTINGS_SEGMENTATION_COLORCHECKER_SG = {'aspect_ratio': 1.4, 'aspect_ratio_maximum': 1.54, 'aspect_ratio_minimum': 1.26, 'interpolation_method': 2, 'reference_values': None, 'swatch_contour_scale': 1.3333333333333333, 'swatch_minimum_area_factor': 200, 'swatches': 140, 'swatches_achromatic_slice': slice(115, 120, 1), 'swatches_chromatic_slice': slice(48, 53, 1), 'swatches_count_maximum': 210, 'swatches_count_minimum': 70, 'swatches_horizontal': 14, 'swatches_vertical': 10, 'transform': {}, 'working_height': 1028, 'working_width': 1440}**

Settings for the segmentation of the *X-Rite ColorChecker SG**.

colour_checker_detection.SETTINGS_SEGMENTATION_COLORCHECKER_NANO

colour_checker_detection.**SETTINGS_SEGMENTATION_COLORCHECKER_NANO = {'aspect_ratio': 1.5, 'aspect_ratio_maximum': 2.0999999999999996, 'aspect_ratio_minimum': 1.0499999999999998, 'interpolation_method': 2, 'reference_values': array([[0.17355167, 0.07874029, 0.05326058], [0.55946176, 0.27734355, 0.21194777], [0.10509124, 0.18955202, 0.32693865], [0.10506442, 0.15021316, 0.05221047], [0.22885963, 0.21350031, 0.42346758], [0.11449231, 0.50663347, 0.41229432], [0.74499115, 0.20172072, 0.0325174], [0.0606182 , 0.10259253, 0.38373146], [0.56055825, 0.08072134, 0.11432307], [0.10983077, 0.04254067, 0.13682661], [0.32967574, 0.49495612, 0.04886544], [0.7689789 , 0.35655545, 0.02534346], [0.0225082 , 0.04870543, 0.28081679], [0.0444356 , 0.29068277, 0.06458335], [0.44636923, 0.03676343, 0.0406788], [0.83803037, 0.57175305, 0.01273052], [0.52392518, 0.07924915, 0.28656418], [-0.04308491, 0.23415773, 0.37506175], [0.87919095, 0.88476747, 0.8349529], [0.58443959, 0.59212352, 0.58458201], [0.35767777, 0.36706043, 0.36528718], [0.19008669, 0.19086038, 0.1898278], [0.08593528, 0.08873843, 0.08978779], [0.03135966, 0.03149993, 0.03231098]]), 'swatch_contour_scale': 1.5, 'swatch_minimum_area_factor': 200, 'swatches': 24, 'swatches_achromatic_slice': slice(19, 23, 1), 'swatches_chromatic_slice': slice(1, 5, 1), 'swatches_count_maximum': 30, 'swatches_count_minimum': 18, 'swatches_horizontal': 6, 'swatches_vertical': 4, 'transform': {}, 'working_height': 960, 'working_width': 1440}**

Settings for the segmentation of the *X-Rite ColorChecker Nano**.

colour_checker_detection.segmenter_default

colour_checker_detection.**segmenter_default**(*image: ArrayLike*, *cctf_encoding: Callable = eotf_inverse_sRGB*, *apply_cctf_encoding: bool = True*, *additional_data: bool = False*, ***kwargs: Any*) → DataSegmentationColourCheckers | NDArrayInt

Detect the colour checker rectangles in given image $image$ using segmentation.

The process is a follows:

- Input image $image$ is converted to a grayscale image $image_g$ and normalised to range [0, 1].
- Image $image_g$ is denoised using multiple bilateral filtering passes into image $image_d$.
- Image $image_d$ is thresholded into image $image_t$.
- Image $image_t$ is eroded and dilated to cleanup remaining noise into image $image_k$.
- Contours are detected on image $image_k$
- Contours are filtered to only keep squares/swatches above and below defined surface area.
- Squares/swatches are clustered to isolate region-of-interest that are potentially colour checkers: Contours are scaled by a third so that colour checkers swatches are joined, creating a large rectangular cluster. Rectangles are fitted to the clusters.
- Clusters with an aspect ratio different to the expected one are rejected, a side-effect is that the complementary pane of the *X-Rite ColorChecker Passport* is omitted.
- Clusters with a number of swatches close to the expected one are kept.

Parameters

- **image** (ArrayLike) – Image to detect the colour checker rectangles from.
- **cctf_encoding** (Callable) – Encoding colour component transfer function / opto-electronic transfer function used when converting the image from float to 8-bit.
- **apply_cctf_encoding** (bool) – Apply the encoding colour component transfer function / opto-electronic transfer function.
- **additional_data** (bool) – Whether to output additional data.
- **adaptive_threshold_kwargs** – Keyword arguments for cv2.adaptiveThreshold() definition.
- **aspect_ratio** – Colour checker aspect ratio, e.g. 1.5.
- **aspect_ratio_minimum** – Minimum colour checker aspect ratio for detection: projective geometry might reduce the colour checker aspect ratio.
- **aspect_ratio_maximum** – Maximum colour checker aspect ratio for detection: projective geometry might increase the colour checker aspect ratio.
- **bilateral_filter_iterations** – Number of iterations to use for bilateral filtering.
- **bilateral_filter_kwargs** – Keyword arguments for cv2.bilateralFilter() definition.
- **convolution_iterations** – Number of iterations to use for the erosion / dilation process.
- **convolution_kernel** – Convolution kernel to use for the erosion / dilation process.
- **interpolation_method** – Interpolation method used when resizing the images, *cv2.INTER_CUBIC* and *cv2.INTER_LINEAR* methods are recommended.

- **reference_values** – Reference values for the colour checker of interest.
- **swatch_contour_scale** – As the image is filtered, the swatches area will tend to shrink, the generated contours can thus be scaled.
- **swatch_minimum_area_factor** – Swatch minimum area factor f with the minimum area m_a expressed as follows: $m_a = image_w * image_h / s_c / f$ where $image_w$, $image_h$ and s_c are respectively the image width, height and the swatches count.
- **swatches** – Colour checker swatches total count.
- **swatches_achromatic_slice** – A *slice* instance defining achromatic swatches used to detect if the colour checker is upside down.
- **swatches_chromatic_slice** – A *slice* instance defining chromatic swatches used to detect if the colour checker is upside down.
- **swatches_count_maximum** – Maximum swatches count to be considered for the detection.
- **swatches_count_minimum** – Minimum swatches count to be considered for the detection.
- **swatches_horizontal** – Colour checker swatches horizontal columns count.
- **swatches_vertical** – Colour checker swatches vertical row count.
- **transform** – Transform to apply to the colour checker image post-detection.
- **working_width** – Width the input image is resized to for detection.
- **working_height** – Height the input image is resized to for detection.
- **kwargs** (Any) –

Returns
Colour checker rectangles and additional data or colour checker rectangles only.

Return type
colour_checker_detection.DataSegmentationColourCheckers or np.ndarray

Notes

- Multiple colour checkers can be detected if present in image.

Examples

```
>>> import os
>>> from colour import read_image
>>> from colour_checker_detection import ROOT_RESOURCES_TESTS
>>> path = os.path.join(
...     ROOT_RESOURCES_TESTS,
...     "colour_checker_detection",
...     "detection",
...     "IMG_1967.png",
... )
>>> image = read_image(path)
>>> segmenter_default(image)
array([[[ 358,  691],
        [ 373,  219],
        [1086,  242],
        [1071,  713]]]...)
```

colour_checker_detection.detect_colour_checkers_segmentation

colour_checker_detection.**detect_colour_checkers_segmentation**(*image: str | ArrayLike*, *samples: int = 32*, *cctf_decoding: Callable = eotf_sRGB*, *apply_cctf_decoding: bool = False*, *segmenter: Callable = segmenter_default*, *segmenter_kwargs: dict | None = None*, *show: bool = False*, *additional_data: bool = False*, ***kwargs: Any*) → Tuple[DataDetectionColourChecker | NDArrayFloat, ...]

Detect the colour checkers swatches in given image using segmentation.

Parameters

- **image** (str | ArrayLike) – Image (or image path to read the image from) to detect the colour checkers swatches from.
- **samples** (int) – Sample count to use to average (mean) the swatches colours. The effective sample count is $samples^2$.
- **cctf_decoding** (Callable) – Decoding colour component transfer function / opto-electronic transfer function used when converting the image from 8-bit to float.
- **apply_cctf_decoding** (bool) – Apply the decoding colour component transfer function / opto-electronic transfer function.
- **segmenter** (Callable) – Callable responsible to segment the image and extract the colour checker rectangles.
- **segmenter_kwargs** (dict | None) – Keyword arguments to pass to the segmenter.
- **show** (bool) – Whether to show various debug images.
- **additional_data** (bool) – Whether to output additional data.
- **adaptive_threshold_kwargs** – Keyword arguments for cv2.adaptiveThreshold() definition.
- **aspect_ratio** – Colour checker aspect ratio, e.g. 1.5.
- **aspect_ratio_minimum** – Minimum colour checker aspect ratio for detection: projective geometry might reduce the colour checker aspect ratio.
- **aspect_ratio_maximum** – Maximum colour checker aspect ratio for detection: projective geometry might increase the colour checker aspect ratio.
- **bilateral_filter_iterations** – Number of iterations to use for bilateral filtering.
- **bilateral_filter_kwargs** – Keyword arguments for cv2.bilateralFilter() definition.
- **convolution_iterations** – Number of iterations to use for the erosion / dilation process.
- **convolution_kernel** – Convolution kernel to use for the erosion / dilation process.
- **interpolation_method** – Interpolation method used when resizing the images, *cv2.INTER_CUBIC* and *cv2.INTER_LINEAR* methods are recommended.
- **reference_values** – Reference values for the colour checker of interest.

- **swatch_contour_scale** – As the image is filtered, the swatches area will tend to shrink, the generated contours can thus be scaled.
- **swatch_minimum_area_factor** – Swatch minimum area factor f with the minimum area m_a expressed as follows: $m_a = image_w * image_h / s_c / f$ where $image_w$, $image_h$ and s_c are respectively the image width, height and the swatches count.
- **swatches** – Colour checker swatches total count.
- **swatches_achromatic_slice** – A *slice* instance defining achromatic swatches used to detect if the colour checker is upside down.
- **swatches_chromatic_slice** – A *slice* instance defining chromatic swatches used to detect if the colour checker is upside down.
- **swatches_count_maximum** – Maximum swatches count to be considered for the detection.
- **swatches_count_minimum** – Minimum swatches count to be considered for the detection.
- **swatches_horizontal** – Colour checker swatches horizontal columns count.
- **swatches_vertical** – Colour checker swatches vertical row count.
- **transform** – Transform to apply to the colour checker image post-detection.
- **working_width** – Width the input image is resized to for detection.
- **working_height** – Height the input image is resized to for detection.
- **kwargs** (Any) –

Returns
Tuple of DataDetectionColourChecker class instances or colour checkers swatches.

Return type
class`tuple`

Examples

```
>>> import os
>>> from colour import read_image
>>> from colour_checker_detection import ROOT_RESOURCES_TESTS
>>> path = os.path.join(
...     ROOT_RESOURCES_TESTS,
...     "colour_checker_detection",
...     "detection",
...     "IMG_1967.png",
... )
>>> image = read_image(path)
>>> detect_colour_checkers_segmentation(image)
(array([[ 0.360005  ,  0.22310828,  0.11760835],
       [ 0.6258309 ,  0.39448667,  0.24166533],
       [ 0.33198   ,  0.31600377,  0.28866866],
       [ 0.3046006 ,  0.273321  ,  0.10486555],
       [ 0.41751358,  0.31914026,  0.30789137],
       [ 0.34866226,  0.43934596,  0.29126382],
       [ 0.67983997,  0.35236534,  0.06997226],
       [ 0.27118555,  0.25352538,  0.33078724],
       [ 0.62091863,  0.27034152,  0.18652563],
       [ 0.3071613 ,  0.17978874,  0.19181632],
       [ 0.48547146,  0.4585586 ,  0.03294956],
```

(continues on next page)

(continued from previous page)

```
       [ 0.6507678 ,  0.40023172,  0.01607676],
       [ 0.19286253,  0.18585181,  0.27459183],
       [ 0.28054565,  0.38513032,  0.1224441 ],
       [ 0.5545431 ,  0.21436104,  0.12549178],
       [ 0.72068894,  0.51493925,  0.00548734],
       [ 0.5772921 ,  0.2577179 ,  0.2685553 ],
       [ 0.17289193,  0.3163792 ,  0.2950853 ],
       [ 0.7394083 ,  0.60953134,  0.4383072 ],
       [ 0.6281671 ,  0.51759964,  0.37215686],
       [ 0.51360977,  0.42048824,  0.2985709 ],
       [ 0.36953217,  0.30218402,  0.20827036],
       [ 0.26286703,  0.21493268,  0.14277342],
       [ 0.16102524,  0.13381621,  0.08047409]]...),)
```

3.1.2 Indices and tables

- genindex
- search

CHAPTER

FOUR

1.4 CODE OF CONDUCT

The *Code of Conduct*, adapted from the Contributor Covenant 1.4, is available on the Code of Conduct page.

CHAPTER

FIVE

1.5 CONTACT & SOCIAL

The *Colour Developers* can be reached via different means:

- Email
- Facebook
- Github Discussions
- Gitter
- Twitter

CHAPTER

SIX

1.6 ABOUT

Colour - Checker Detection by Colour Developers

https://github.com/colour-science/colour-checker-detection

BIBLIOGRAPHY

[Abe11] Felix Abecassis. Opencv - rotation (deskewing). 2011. URL: http://felix.abecassis.me/2011/10/opencv-rotation-deskewing/ (visited on 2018-10-27).

INDEX

D

I

S