# Caer

***Release '2.0.3'***

**Jason Dsouza**

**May 29, 2023**

# FIRST STEPS

Caer is a lightweight Computer Vision library for high-performance AI research. It simplifies your approach towards Computer Vision by abstracting away unnecessary boilerplate code enabling maximum flexibility. By offering powerful image and video processing algorithms, Caer provides both casual and advanced users with an elegant interface for Machine vision operations.

It leverages the power of libraries like OpenCV and Pillow to speed up your Computer Vision workflow — making it fully compatible with other frameworks such as PyTorch and Tensorflow.

This design philosophy makes Caer ideal for students, researchers, hobbyists and even experts in the fields of Deep Learning and Computer Vision to quickly prototype deep learning models or research ideas.

CHAPTER

ONE

# INTRODUCTION

CHAPTER

TWO

# INSTALLATION & SETUP

Caer supports an installation of Python 3.6 above, available on Windows, MacOS and Linux systems.

## 2.1 Version check

To see whether `caer` is already installed or to check if an install has worked, run the following in a Python shell or Jupyter notebook:

```
>> import caer
>> print(caer.__version__)
```

or, from the command line:

```
python -c "import caer; print(caer.__version__)"
```

(Try `python3` if `python` is unsuccessful.)

You'll see the version number if `caer` is installed and an error message otherwise.

## 2.2 Installation

### 2.2.1 pip (Recommended)

Prerequisites to a pip install: You are able to use your system's command line to install packages and are using a virtual environment (any of several).

To install the current `caer` you'll need at least Python 3.6.1. If you have an older version of Python, you will not be able to use `caer`.

```
$ pip install --upgrade caer
```

Alternatively, you may download the wheels from PyPi

### 2.2.2 Warning

Do not use the commands `sudo` and `pip` together as `pip` may overwrite critical system libraries which may require you to reinstall your operating system.

### 2.2.3 Bleeding Edge

If a bug fix was made in the repo and you can't wait till a new release is made, you can install the bleeding edge version of `caer` using:

```
pip install git+https://github.com/jasmcaus/caer.git
```

### 2.2.4 From Source

If you plan to develop `caer` yourself, or want to be on the cutting edge, you can use an editable install:

First, uninstall any existing installations:

```
pip uninstall -y caer
```

Clone the repo:

```
git clone https://github.com/jasmcaus/caer.git
cd caer
pip install -e . # Do this once to add the package to the Python Path
```

To update the installation:

```
git pull  # Grabs the latest source
pip install -e . # Reinstalls Caer
```

## 2.3 System package managers

Using a package manager (`yum`, `apt-get`, etc.) to install `caer` or other Python packages is not your best option:

- You're likely to get an older version.
- You'll probably want to make updates and add new packages outside of the package manager, leaving you with the same kind ofdependency conflicts you see when using pip without a virtual environment.
- There's an added risk because operating systems use Python, so if you make system-wide Python changes (installing as root or using sudo), you can break the operating system.

CHAPTER

THREE

# TYPE CHECKING

## 3.1 Static type checking and real-time linting for Vision code.

By this point, you've probably seen that `Caer` uses the new **Python 3.6+** syntax for type hints (also known as *type annotations*). The entire code base is type-annotated and it is recommended that you add *at least some types* to your own code, too!

Type annotations can make your numeric code much more explicit, making it (much) easier to debug. They also allow your editor (and other tools) to perform type checks before executing your code. For example, if you try to add a `str` and an `int`, your editor will probably warn you that it is an invalid operation without having to wait until you run the invalid code. It may also tell you that a function expects a `float`, so you don't pass it an invalid type. If your layer is typed as `caer.Tensor`, Caer can warn you if its inputs and outputs are incompatible with the rest of your Vision model.

Our's type-system won't catch every error: it has no representation for the sizes of your `Tensor` dimensions, so a lot of invalid operations can't be detected until runtime. Sometimes the syntax gets quite ugly, and the error messages are often frustratingly opaque. Nevertheless, we do recommend you try it out, especially for your model definitions and the functions you end up using.

## 3.2 Installation & Setup

`mypy` is the *standard* type-checker for Python. You can install it via `pip` or `conda`. If you use a virtual environment for your project, ensure that you install it in the same environment.

```
$ pip install mypy
```

or

```
$ conda install -c conda-forge mypy
```

We are working on a `mypy` plugin that will extend the normal functionality of `mypy` to perform additional type checks in code using Caer. If you installed Caer, you already have the plugin. To enable the Caer plugin for `mypy`, you just have to create a file `mypy.ini` at the *root* of your project folder. This will tell `mypy` to use the plugin in the module `caer.mypy`. If you use `pydantic` for advanced configuration, you can also enable `pydantic`'s plugin. If you're using Caer as part of your Python package, you can also add the `[mypy]` section to your package's `setup.cfg`.

```
[mypy]
plugins = caer.mypy
```

To type check a file or directory, you can now use the `mypy` command:

```
$ mypy file.py
```

## 3.3 Setting up linting in your editor

Real-time linting is especially powerful, as it lets you type-check your code as it leaves your fingers. This often lets you catch errors in their original context, when they're least confusing. It can also save you trips to the documentation.

If you use Visual Studio Code, make sure you install the Python extension. Then select the appropriate environment in your editor. If you installed mypy in the same environment and select it in your editor, after adding the mypy.ini file (as described above) everything should work.

For PyCharm users, make sure you configure the Python Interpreter for your project. Then install the "Mypy" plugin. You may also want to install the "Mypy (Official)" plugin. If you installed mypy in the same environment/interpreter, after adding the mypy.ini file (as described above) and installing the plugin, everything should work.

For other editors, the `mypy` docs has instructions for editors like Vim, Emacs, Sublime Text and Atom.

## 3.4 Static Type Checking

*Static type checking* means that your editor (or other tools) will check the code using the declared types before running it. Because it is done before running the code, it's called "static". The contrary would be *dynamic* type checking, where checks are performed at runtime, while the program is running and the code is being executed. (Once complete, Caer will also do runtime validation!) As editors and similar tools can't just randomly run your code to verify that it's correct, we have these type annotations to help editors check the code and provide autocompletion.

Even if you never run a type-checker, adding type-annotations to your code can greatly improve its readability. Multi-dimensional Tensor libraries like `coreten` and `numpy` make it easy to write terse, fairly general code; but when you revisit the code later, it's often very hard to figure out what's happening without executing the code and debugging.

Consider this function (from the `caer.transforms` module):

```
def darken(img: Tensor, coeff: float) -> Tensor:
    ...
    return darkened
```

over

```
def darken(img, coeff):
    ...
    return darkened
```

Type annotations provide a relatively concise way to document some of the most important information about your code. The same information can be provided in comments, but unless you use consistent syntax, your type comments will probably be much longer and more distracting than the equivalent annotations.

Another advantage of type annotations as documentation is that they can be queried for more detail, while with comments, you have to choose the level of detail to provide up-front. Thinc's type annotations take into account numpy's tricky indexing system, and also the semantics of the different reduction operations as different arguments are passed in. This makes it much easier to follow along with steps that might have felt obvious to the author of the code.

CHAPTER

FOUR

# CONTRIBUTING TO CAER

Thank you for choosing to contribute to Caer. We'd love to accept your patches! For those just getting started, Github has a how to. All bug fixes, new functionality, new tutorials etc. are (and should be) submitted via GitHub's mechanism of pull requests.

Caer was made publically available in August 2020 by Jason Dsouza <jasmcaus@gmail.com> and we are proud to have over 1.2m installs since then! Caer is now maintained by awesome community members just like you.

We have written a comprehensive Contribution Guide that you can follow when contributing to the project.

## 4.1 Documentation

There is always room for improvement in documentation. We welcome all pull requests to fix typos/improve grammar or semantic structuring of documents. Here are a few documents you can work on:

- Official Tutorials
- README

## 4.2 Open Issues

If you would like to help in working on open issues. Look out for the following tags: `good first issue`, `help wanted`, `open for contribution`

## 4.3 Major Contributions

If you are willing to make a major contribution you can always lookout for the active sprint under `Projects` and discuss the proposal with one of our maintainers.

## 4.4 What we currently need help on

- Improving unit-test cases and test coverage
- Include GPU support

## 4.5 If you write articles:

If you are interested or have already written a story covering `Caer`, you can submit your story in `markdown` format as a PR here. To convert medium stories into `markdown` format, you may use this chrome extension

CHAPTER

FIVE

# CAER.COLOR

## 5.1 Using the `to_` syntax

### 5.1.1 to_rgb

caer.color.**to_rgb**(*tens*)

Converts any supported colorspace to RGB

**Parameters** **tens** (*Tensor*) –

**Return type** Tensor

**Returns** Tensor

### 5.1.2 to_bgr

caer.color.**to_bgr**(*tens*)

Converts any supported colorspace to BGR

**Parameters** **tens** (*Tensor*) –

**Return type** Tensor

**Returns** Tensor

### 5.1.3 to_gray

caer.color.**to_gray**(*tens*)

Converts any supported colorspace to grayscale

**Parameters** **tens** (*Tensor*) –

**Return type** Tensor

**Returns** Tensor

### 5.1.4 to_hsv

caer.color.**to_hsv**(*tens*)

Converts any supported colorspace to hsv

**Parameters** **tens** (*Tensor*) –

**Return type** Tensor

**Returns** Tensor

### 5.1.5 to_hls

caer.color.**to_hls**(*tens*)

Converts any supported colorspace to HLS

**Parameters** **tens** (*Tensor*) –

**Return type** Tensor

**Returns** Tensor

### 5.1.6 to_lab

caer.color.**to_lab**(*tens*)

Converts any supported colorspace to LAB

**Parameters** **tens** (*Tensor*) –

**Return type** Tensor

**Returns** Tensor

## 5.2 From the RGB colorspace

### 5.2.1 rgb2bgr

caer.color.**rgb2bgr**(*tens*)

Converts an RGB Tensor to its BGR version.

**Parameters** **tens** (*Tensor*) – Valid RGB Tensor

**Return type** Tensor

**Returns** BGR Tensor of shape `(height, width, channels)`

**Raises** **ValueError** – If *tens* is not of shape 3

### 5.2.2 rgb2gray

caer.color.**rgb2gray**(*tens*)

Converts an RGB Tensor to its Grayscale version.

**Parameters** **tens** (*Tensor*) – Valid RGB Tensor

**Return type** Tensor

**Returns** Grayscale Tensor of shape (height, width, channels)

**Raises** **ValueError** – If *tens* is not of shape 3

### 5.2.3 rgb2hsv

caer.color.**rgb2hsv**(*tens*)

Converts an RGB Tensor to its HSV version.

**Parameters** **tens** (*Tensor*) – Valid RGB Tensor

**Return type** Tensor

**Returns** HSV Tensor of shape (height, width, channels)

**Raises** **ValueError** – If *tens* is not of shape 3

### 5.2.4 rgb2lab

caer.color.**rgb2bgr**(*tens*)

Converts an RGB Tensor to its BGR version.

**Parameters** **tens** (*Tensor*) – Valid RGB Tensor

**Return type** Tensor

**Returns** BGR Tensor of shape (height, width, channels)

**Raises** **ValueError** – If *tens* is not of shape 3

---

## 5.3 From the BGR colorspace

### 5.3.1 bgr2rgb

caer.color.**bgr2rgb**(*tens*)

Converts a BGR Tensor to its RGB version.

**Parameters** **tens** (*Tensor*) – Valid BGR Tensor

**Return type** Tensor

**Returns** RGB Tensor of shape `(height, width, channels)`

**Raises** `ValueError` – If *tens* is not of shape 3

### 5.3.2 bgr2gray

`caer.color.bgr2gray(tens)`

Converts a BGR Tensor to its Grayscale version.

**Parameters** `tens` (*Tensor*) – Valid BGR Tensor

**Return type** `Tensor`

**Returns** Grayscale Tensor of shape `(height, width, channels)`

**Raises** `ValueError` – If *tens* is not of shape 3

### 5.3.3 bgr2hsv

`caer.color.bgr2hsv(tens)`

Converts a BGR Tensor to its HSV version.

**Parameters** `tens` (*Tensor*) – Valid BGR Tensor

**Return type** `Tensor`

**Returns** HSV Tensor of shape `(height, width, channels)`

**Raises** `ValueError` – If *tens* is not of shape 3

### 5.3.4 bgr2lab

`caer.color.bgr2lab(tens)`

Converts a BGR Tensor to its LAB version.

**Parameters** `tens` (*Tensor*) – Valid BGR Tensor

**Return type** `Tensor`

**Returns** LAB Tensor of shape `(height, width, channels)`

**Raises** `ValueError` – If *tens* is not of shape 3

## 5.4 From the HSV colorspace

### 5.4.1 hsv2rgb

caer.color.**hsv2rgb**(*tens*)

Converts a HSV Tensor to its RGB version.

**Parameters** **tens** (*Tensor*) – Valid HSV Tensor

**Return type** Tensor

**Returns** RGB Tensor of shape (height, width, channels)

**Raises** **ValueError** – If *tens* is not of shape 3

### 5.4.2 hsv2bgr

caer.color.**hsv2bgr**(*tens*)

Converts a HSV Tensor to its BGR version.

**Parameters** **tens** (*Tensor*) – Valid HSV Tensor

**Return type** Tensor

**Returns** BGR Tensor of shape (height, width, channels)

**Raises** **ValueError** – If *tens* is not of shape 3

### 5.4.3 hsv2gray

caer.color.**hsv2gray**(*tens*)

Converts a HSV Tensor to its Grayscale version.

**Parameters** **tens** (*Tensor*) – Valid HSV Tensor

**Return type** Tensor

**Returns** Grayscale Tensor of shape (height, width, channels)

**Raises** **ValueError** – If *tens* is not of shape 3

### 5.4.4 hsv2lab

caer.color.**hsv2lab**(*tens*)

Converts a HSV Tensor to its LAB version.

**Parameters** **tens** (*Tensor*) – Valid HSV Tensor

**Return type** Tensor

**Returns** LAB Tensor of shape (height, width, channels)

**Raises** **ValueError** – If *tens* is not of shape 3

## 5.5 From the LAB colorspace

### 5.5.1 lab2rgb

caer.color.**lab2rgb**(*tens*)

Converts an LAB Tensor to its RGB version.

**Parameters** **tens** (*Tensor*) – Valid LAB Tensor

**Return type** Tensor

**Returns** RGB Tensor of shape (height, width, channels)

**Raises** **ValueError** – If *tens* is not of shape 3

### 5.5.2 lab2bgr

caer.color.**lab2bgr**(*tens*)

Converts an LAB Tensor to its BGR version.

**Parameters** **tens** (*Tensor*) – Valid LAB Tensor

**Return type** Tensor

**Returns** BGR Tensor of shape (height, width, channels)

**Raises** **ValueError** – If *tens* is not of shape 3

### 5.5.3 lab2gray

caer.color.**lab2gray**(*tens*)

Converts an LAB Tensor to its Grayscale version.

**Parameters** **tens** (*Tensor*) – Valid LAB Tensor

**Return type** Tensor

**Returns** Grayscale Tensor of shape (height, width, channels)

**Raises** **ValueError** – If *tens* is not of shape 3

### 5.5.4 lab2hsv

caer.color.**lab2hsv**(*tens*)

Converts an LAB Tensor to its HSV version.

**Parameters** **tens** (*Tensor*) – Valid LAB Tensor

**Return type** Tensor

**Returns** HSV Tensor of shape (height, width, channels)

**Raises** **ValueError** – If *tens* is not of shape 3

CHAPTER

SIX

# CAER.DATA

## 6.1 Standard Test Images

### 6.1.1 audio_mixer

caer.data.**audio_mixer**(*target_size=None*, *rgb=True*)

Returns a standard 640x427 image Tensor (RGB, by default) of an audio mixer.

**Parameters**

- **target_size** (*Optional[tuple]*) – Intended target size (follows the `(width, height)` format). If None, the unaltered tensor will be returned.
- **rgb** (*bool*) – Boolean whether to return an RGB Tensor (default is `True`).

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.audio_mixer()
>> tens.shape
(427, 640, 3)
```

### 6.1.2 bear

caer.data.**bear**(*target_size=None*, *rgb=True*)

Returns a standard 640x427 image Tensor (RGB, by default) of a bear.

**Parameters**

- **target_size** (*Optional[tuple]*) – Intended target size (follows the `(width, height)` format). If None, the unaltered tensor will be returned.
- **rgb** (*bool*) – Boolean whether to return an RGB Tensor (default is `True`).

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.bear()
>> tens.shape
(427, 640, 3)
```

### 6.1.3 beverages

caer.data.**beverages**(*target_size=None*, *rgb=True*)

Returns a standard 640x427 image Tensor (RGB, by default) of beverages.

**Parameters**

- **target_size** (*Optional[tuple]*) – Intended target size (follows the `(width, height)` format). If None, the unaltered tensor will be returned.
- **rgb** (*bool*) – Boolean whether to return an RGB Tensor (default is `True`).

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.beverages()
>> tens.shape
(427, 640, 3)
```

### 6.1.4 black_cat

caer.data.**black_cat**(*target_size=None*, *rgb=True*)

Returns a standard 640x427 image Tensor (RGB, by default) of a black cat.

**Parameters**

- **target_size** (*Optional[tuple]*) – Intended target size (follows the `(width, height)` format). If None, the unaltered tensor will be returned.
- **rgb** (*bool*) – Boolean whether to return an RGB Tensor (default is `True`).

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.black_cat()
>> tens.shape
(427, 640, 3)
```

### 6.1.5 blue_tang

caer.data.**blue_tang**(*target_size=None*, *rgb=True*)

Returns a standard 640x414 image Tensor (RGB, by default) of a blue tang (a type of fish).

**Parameters**

- **target_size** (*Optional[tuple]*) – Intended target size (follows the `(width, height)` format). If None, the unaltered tensor will be returned.
- **rgb** (*bool*) – Boolean whether to return an RGB Tensor (default is `True`).

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.audio_mixer()
>> tens.shape
(414, 640, 3)
```

### 6.1.6 camera

caer.data.**camera**(*target_size=None*, *rgb=True*)

Returns a standard 640x427 image Tensor (RGB, by default) of a camera.

**Parameters**

- **target_size** (*Optional[tuple]*) – Intended target size (follows the `(width, height)` format). If None, the unaltered tensor will be returned.
- **rgb** (*bool*) – Boolean whether to return an RGB Tensor (default is `True`).

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.camera()
>> tens.shape
(427, 640, 3)
```

### 6.1.7 controller

caer.data.**controller**(*target_size=None*, *rgb=True*)

Returns a standard 640x427 image Tensor (RGB, by default) of a game controller.

**Parameters**

- **target_size** (*Optional[tuple]*) – Intended target size (follows the `(width, height)` format). If None, the unaltered tensor will be returned.

- **rgb** (*bool*) – Boolean whether to return an RGB Tensor (default is `True`).

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.controller()
>> tens.shape
(427, 640, 3)
```

### 6.1.8 drone

caer.data.**drone**(*target_size=None*, *rgb=True*)

Returns a standard 640x358 image Tensor (RGB, by default) of a robotic drone.

**Parameters**

- **target_size** (*Optional[tuple]*) – Intended target size (follows the `(width, height)` format). If None, the unaltered tensor will be returned.
- **rgb** (*bool*) – Boolean whether to return an RGB Tensor (default is `True`).

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.drone()
>> tens.shape
(358, 640, 3)
```

### 6.1.9 dusk

caer.data.**dusk**(*target_size=None*, *rgb=True*)

Returns a standard 640x427 image Tensor (RGB, by default) of a dusk landscape.

**Parameters**

- **target_size** (*Optional[tuple]*) – Intended target size (follows the `(width, height)` format). If None, the unaltered tensor will be returned.
- **rgb** (*bool*) – Boolean whether to return an RGB Tensor (default is `True`).

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.dusk()
>> tens.shape
(427, 640, 3)
```

### 6.1.10 fighter_fish

caer.data.**fighter_fish**(*target_size=None*, *rgb=True*)

Returns a standard 640x640 image Tensor (RGB, by default) of a fighter fish.

**Parameters**

- **target_size** (*Optional[tuple]*) – Intended target size (follows the `(width, height)` format). If None, the unaltered tensor will be returned.
- **rgb** (*bool*) – Boolean whether to return an RGB Tensor (default is `True`).

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.fighter_fish()
>> tens.shape
(640, 640, 3)
```

### 6.1.11 gold_fish

caer.data.**gold_fish**(*target_size=None*, *rgb=True*)

Returns a standard 640x901 image Tensor (RGB, by default) of a gold fish.

**Parameters**

- **target_size** (*Optional[tuple]*) – Intended target size (follows the `(width, height)` format). If None, the unaltered tensor will be returned.
- **rgb** (*bool*) – Boolean whether to return an RGB Tensor (default is `True`).

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.gold_fish()
>> tens.shape
(901, 640, 3)
```

### 6.1.12 green_controller

caer.data.**green_controller**(*target_size=None*, *rgb=True*)

Returns a standard 640x512 image Tensor (RGB, by default) of a green game controller.

**Parameters**

- **target_size** (*Optional[tuple]*) – Intended target size (follows the `(width, height)` format). If None, the unaltered tensor will be returned.

- **rgb** (*bool*) – Boolean whether to return an RGB Tensor (default is `True`).

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.green_controller()
>> tens.shape
(512, 640, 3)
```

### 6.1.13 green_fish

caer.data.**green_fish**(*target_size=None*, *rgb=True*)

Returns a standard 640x430 image Tensor (RGB, by default) of a green fish.

**Parameters**

- **target_size** (*Optional[tuple]*) – Intended target size (follows the `(width, height)` format). If None, the unaltered tensor will be returned.
- **rgb** (*bool*) – Boolean whether to return an RGB Tensor (default is `True`).

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.green_fish()
>> tens.shape
(430, 640, 3)
```

### 6.1.14 guitar

caer.data.**guitar**(*target_size=None*, *rgb=True*)

Returns a standard 640x427 image Tensor (RGB, by default) of a guitar.

**Parameters**

- **target_size** (*Optional[tuple]*) – Intended target size (follows the `(width, height)` format). If None, the unaltered tensor will be returned.
- **rgb** (*bool*) – Boolean whether to return an RGB Tensor (default is `True`).

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.guitar()
>> tens.shape
(427, 640, 3)
```

### 6.1.15 island

caer.data.**island**(*target_size=None*, *rgb=True*)

Returns a standard 640x426 image Tensor (RGB, by default) of an island.

**Parameters**

- **target_size** (*Optional[tuple]*) – Intended target size (follows the `(width, height)` format). If None, the unaltered tensor will be returned.
- **rgb** (*bool*) – Boolean whether to return an RGB Tensor (default is `True`).

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.island()
>> tens.shape
(426, 640, 3)
```

### 6.1.16 jellyfish

caer.data.**jellyfish**(*target_size=None*, *rgb=True*)

Returns a standard 640x427 image Tensor (RGB, by default) of a jellyfish.

**Parameters**

- **target_size** (*Optional[tuple]*) – Intended target size (follows the `(width, height)` format). If None, the unaltered tensor will be returned.
- **rgb** (*bool*) – Boolean whether to return an RGB Tensor (default is `True`).

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.jellyfish()
>> tens.shape
(427, 640, 3)
```

### 6.1.17 laptop

caer.data.**laptop**(*target_size=None*, *rgb=True*)

Returns a standard 640x427 image Tensor (RGB, by default) of a laptop.

**Parameters**

- **target_size** (*Optional[tuple]*) – Intended target size (follows the `(width, height)` format). If None, the unaltered tensor will be returned.

- **rgb** (*bool*) – Boolean whether to return an RGB Tensor (default is `True`).

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.laptop()
>> tens.shape
(427, 640, 3)
```

### 6.1.18 mountain

`caer.data.`**mountain**(*target_size=None*, *rgb=True*)

Returns a standard 640x427 image Tensor (RGB, by default) of a mountain.

**Parameters**

- **target_size** (*Optional[tuple]*) – Intended target size (follows the `(width, height)` format). If None, the unaltered tensor will be returned.
- **rgb** (*bool*) – Boolean whether to return an RGB Tensor (default is `True`).

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.mountain()
>> tens.shape
(427, 640, 3)
```

### 6.1.19 night

`caer.data.`**night**(*target_size=None*, *rgb=True*)

Returns a standard 640x427 image Tensor (RGB, by default) of a night landscape.

**Parameters**

- **target_size** (*Optional[tuple]*) – Intended target size (follows the `(width, height)` format). If None, the unaltered tensor will be returned.
- **rgb** (*bool*) – Boolean whether to return an RGB Tensor (default is `True`).

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.night()
>> tens.shape
(427, 640, 3)
```

### 6.1.20 puppies

caer.data.**puppies**(*target_size=None*, *rgb=True*)

Returns a standard 640x427 image Tensor (RGB, by default) of a litter of puppies.

**Parameters**

- **target_size** (*Optional[tuple]*) – Intended target size (follows the `(width, height)` format). If None, the unaltered tensor will be returned.
- **rgb** (*bool*) – Boolean whether to return an RGB Tensor (default is `True`).

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.puppies()
>> tens.shape
(427, 640, 3)
```

### 6.1.21 puppy

caer.data.**puppy**(*target_size=None*, *rgb=True*)

Returns a standard 640x512 image Tensor (RGB, by default) of a puppy.

**Parameters**

- **target_size** (*Optional[tuple]*) – Intended target size (follows the `(width, height)` format). If None, the unaltered tensor will be returned.
- **rgb** (*bool*) – Boolean whether to return an RGB Tensor (default is `True`).

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.puppy()
>> tens.shape
(512, 640, 3)
```

### 6.1.22 red_fish

caer.data.**red_fish**(*target_size=None*, *rgb=True*)

Returns a standard 640x427 image Tensor (RGB, by default) of a red fish.

**Parameters**

- **target_size** (*Optional[tuple]*) – Intended target size (follows the `(width, height)` format). If None, the unaltered tensor will be returned.

- **rgb** (*bool*) – Boolean whether to return an RGB Tensor (default is `True`).

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.red_fish()
>> tens.shape
(427, 640, 3)
```

### 6.1.23 phone

`caer.data.`**phone**(*target_size=None*, *rgb=True*)

Returns a standard 640x427 image Tensor (RGB, by default) of a rotary phone.

**Parameters**

- **target_size** (*Optional[tuple]*) – Intended target size (follows the `(width, height)` format). If None, the unaltered tensor will be returned.
- **rgb** (*bool*) – Boolean whether to return an RGB Tensor (default is `True`).

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.phone()
>> tens.shape
(427, 640, 3)
```

### 6.1.24 sea_turtle

`caer.data.`**sea_turtle**(*target_size=None*, *rgb=True*)

Returns a standard 640x400 image Tensor (RGB, by default) of a sea turtle.

**Parameters**

- **target_size** (*Optional[tuple]*) – Intended target size (follows the `(width, height)` format). If None, the unaltered tensor will be returned.
- **rgb** (*bool*) – Boolean whether to return an RGB Tensor (default is `True`).

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.sea_turtle()
>> tens.shape
(400, 640, 3)
```

### 6.1.25 snow

caer.data.**snow**(*target_size=None*, *rgb=True*)

Returns a standard 640x360 image Tensor (RGB, by default) of snow.

**Parameters**

- **target_size** (*Optional[tuple]*) – Intended target size (follows the `(width, height)` format). If None, the unaltered tensor will be returned.
- **rgb** (*bool*) – Boolean whether to return an RGB Tensor (default is `True`).

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.snow()
>> tens.shape
(360, 640, 3)
```

### 6.1.26 sunrise

caer.data.**sunrise**(*target_size=None*, *rgb=True*)

Returns a standard 640x427 image Tensor (RGB, by default) of a sunrise landscape.

**Parameters**

- **target_size** (*Optional[tuple]*) – Intended target size (follows the `(width, height)` format). If None, the unaltered tensor will be returned.
- **rgb** (*bool*) – Boolean whether to return an RGB Tensor (default is `True`).

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.sunrise()
>> tens.shape
(427, 640, 3)
```

### 6.1.27 tent

caer.data.**tent**(*target_size=None*, *rgb=True*)

Returns a standard 640x427 image Tensor (RGB, by default) of a tent.

**Parameters**

- **target_size** (*Optional[tuple]*) – Intended target size (follows the `(width, height)` format). If None, the unaltered tensor will be returned.

- **rgb** (*bool*) – Boolean whether to return an RGB Tensor (default is `True`).

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.tent()
>> tens.shape
(427, 640, 3)
```

CHAPTER

SEVEN

# CAER.IO

## 7.1 Reading Images

### 7.1.1 imread

caer.io.**imread**(*image_path*, *rgb=True*, *target_size=None*, *resize_factor=None*, *preserve_aspect_ratio=False*, *interpolation='bilinear'*)

Loads in an image from *image_path* (can be either a system filepath or a URL)

**Parameters**

- **image_path** (*str*) – Filepath/URL to read the image from.
- **rgb** (*bool*) – Boolean to keep RGB ordering. Default: True

**Return type** Tensor

**Returns** Tensor with shape `(height, width, channels)`.

Examples:

```
>> tens = caer.imread(tens_path) # From FilePath
>> tens.shape
(427, 640, 3)

>> tens = caer.imread('https://raw.githubusercontent.com/jasmcaus/caer/master/caer/
↪data/beverages.jpg') # From URL
>> tens.shape
(427, 640, 3)
```

## 7.2 Saving Images

### 7.2.1 imsave

caer.io.**imsave**(*path*, *tens*)

Saves a Tensor to *path*

**Parameters** **path** (*str*) – Filepath to save the image to

**Returns** `True`; if *tens* was written to *path* `False`; otherwise

Examples:

```
>> tens = caer.data.audio_mixer()
>> caer.imsave('audio_mixer.png', tens)
True
```

**Return type** `bool`

---

## 7.3 Resizing Images

### 7.3.1 imread

`caer.io.resize`(*tens*, *target_size=None*, *resize_factor=None*, *preserve_aspect_ratio=False*, *interpolation='bilinear'*)

Resizes an image to a target_size without aspect ratio distortion.

Your output images will be of size `target_size`, and will not be distorted. Instead, the parts of the image that do not fit within the target size get cropped out.

The resizing process is: 1. Resize the image as minimally as possible. 2. Take the largest centered crop of the image with dimensions = `target_size`.

Alternatively, you may use: `` `python size = (200,200) tens = caer.resize(tens, target_size=size, preserve_aspect_ratio=True) ` ``

---

**Note:** `caer.imread()` comes with an in-built functionality to resize your images, eliminating the need for you to call `caer.resize()`. This is purely optional and may appeal to certain users.

You may also use `caer.smart_resize()` for on-the-fly image resizing that *preserves the aspect ratio*.

---

**Parameters**

- **tens** (*Tensor*) – Input Image. Must be in the format `(height, width, channels)`.
- **target_size** (*tuple*) – Target size. Must be a tuple of `(width, height)` integer.
- **resize_factor** (*float, tuple*) – Resizing Factor to employ. Shrinks the image if `resize_factor < 1` Enlarges the image if `resize_factor > 1`
- **preserve_aspect_ratio** (*bool*) – Prevent aspect ratio distortion (employs center crop).
- **interpolation** (*str*) – Interpolation to use for resizing. Defaults to *'bilinear'*. Supports *'bilinear'*, *'bicubic'*, *'area'*, *'nearest'*.

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.sunrise()
>> tens.shape
(427, 640, 3)

>> resized = caer.resize(tens, target_size=(200,200)) # Hard-resize. May distort␣
↪aspect ratio
>> resized.shape
(200, 200, 3)

>> resized_wf = caer.resize(tens, resize_factor=.5) # Resizes the image to half its␣
↪dimensions
>> resized_wf.shape
(213, 320, 3)

>> resized = caer.resize(tens, target_size=(200,200), preserve_aspect_ratio=True) #␣
↪Preserves aspect ratio
>> resized.shape
(200, 200, 3)
```

## 7.4 Smart Resizing

### 7.4.1 smart_resize

caer.io.**smart_resize**(*tens*, *target_size*, *interpolation='bilinear'*)

Resizes an image to a target_size without aspect ratio distortion.

Your output images will be of size *target_size*, and will not be distorted. Instead, the parts of the image that do not fit within the target size get cropped out.

The resizing process is: 1. Resize the image as minimally as possible. 2. Take the largest centered crop of the image with dimensions = *target_size*.

Alternatively, you may use: `python size = (200,200) tens = caer.resize(tens, target_size=size, preserve_aspect_ratio=True) `

**Parameters**

- **tens** (*Tensor*) – Input Image. Must be in the format *(height, width, channels)*.
- **target_size** (*tuple*) – Target size. Must be a tuple of *(width, height)* integer.
- **interpolation** (*str*) – Interpolation to use for resizing. Defaults to *'bilinear'*. Supports *'bilinear'*, *'bicubic'*, *'area'*, *'nearest'*.

**Returns** Tensor of shape *(height, width, channels)*

Examples:

```
>> tens = caer.data.sunrise()
>> tens.shape
(427, 640, 3)
>> resized = caer.smart_resize(tens, target_size=(200,200))
>> resized.shape
(200, 200, 3)
```

CHAPTER

# EIGHT

# CAER.PATH

caer.path.**abspath**(*file_name*)

Returns the absolute path of *file_name*

**Return type** str

caer.path.**chdir**(*path*)

Checks into directory *path*

**Return type** None

caer.path.**cwd**()

Returns the filepath to the current working directory

**Return type** str

caer.path.**dirname**(*file*)

Returns the base directory name of *file*

**Return type** str

caer.path.**exists**(*path*)

Checks if a given filepath is valid

**Parameters** **path** (*str*) – Filepath to check

**Return type** bool

**Returns** True; if *path* is a valid filepath False; otherwise

caer.path.**get_size**(*file*, *disp_format='bytes'*)

Returns: the size of *file* in bytes/kb/mb/gb/tb

**Parameters**

- **file** (*str*) – Filepath to check
- **disp_format** (*str*) – Size format (bytes/kb/mb/gb/tb)

**Returns** File size in bytes/kb/mb/gb/tb

**Return type** size (str)

caer.path.**is_image**(*path*)

Checks if a given path is that of a valid image file

**Parameters** **path** (*str*) – Filepath to check

**Return type** bool

**Returns** True; if *path* is a valid image filepath False; otherwise

caer.path.**is_video**(*path*)

Checks if a given path is that of a valid image file

**Parameters** **path** (*str*) – Filepath to check

**Return type** bool

**Returns** True; if *path* is a valid image filepath False; otherwise

caer.path.**isfile**(*path*)

Checks if a given filepath is a valid path

**Parameters** **path** (*str*) – Filepath to check

**Return type** bool

**Returns** True; if *path* is a valid filepath False; otherwise

caer.path.**join**(**paths*)

Join multiple filepaths together

**Return type** str

caer.path.**list_images**(*DIR*, *recursive=True*, *use_fullpath=False*, *show_size=False*, *verbose=True*)

Lists all image files within a specific directory (and sub-directories if *recursive=True*)

**Parameters**

- **DIR** (*str*) – Directory to search for image files
- **recursive** (*bool*) – Indicate whether to search all subdirectories as well (default = False)
- **use_fullpath** (*bool*) – Include full filepaths in the returned list (default = False)
- **show_size** (*bool*) – Prints the disk size of the image files (default = False)

**Returns** List of names (or full filepaths if *use_fullpath=True*) of the image files

**Return type** image_files (list)

caer.path.**list_videos**(*DIR*, *recursive=True*, *use_fullpath=False*, *show_size=False*, *verbose=True*)

Lists all video files within a specific directory (and sub-directories if *recursive=True*)

**Parameters**

- **DIR** (*str*) – Directory to search for video files
- **recursive** (*bool*) – Indicate whether to search all subdirectories as well (default = False)

- **use_fullpath** (*bool*) – Include full filepaths in the returned list (default = False)
- **show_size** (*bool*) – Prints the disk size of the video files (default = False)

**Returns** List of names (or full filepaths if *use_fullpath=True*) of the video files

**Return type** video_files (list)

caer.path.**listdir**(*DIR*, *recursive=False*, *use_fullpath=False*, *ext=None*, *show_size=False*, *verbose=True*)

Lists all files within a specific directory (and sub-directories if *recursive=True*). This can be filtered for certain extensions (by populating *ext*)

**Parameters**

- **DIR** (*str*) – Directory to search for files
- **recursive** (*bool*) – Indicate whether to search all subdirectories as well (default = False)
- **use_fullpath** (*bool*) – Include full filepaths in the returned list (default = False)
- **ext** (*str, list(str), tuple(str)*) – Filter by extension names.
- **show_size** (*bool*) – Prints the disk size of the files (default = False)
- **verbose** (*bool*) – Print info

**Returns** List of names (or full filepaths if *use_fullpath=True*) of the files

**Return type** files (list)

caer.path.**mkdir**(*path*)

Creates a directory at *path*

**Return type** None

CHAPTER

NINE

# CAER.TRANSFORMS

caer.transforms consists of position and color-based transforms for use.

caer.transforms.**adjust_brightness**(*tens*, *coeff*, *rgb=True*)

Adjust the brightness of an image.

**Parameters**

- **tens** (*Tensor*) – Any regular `caer.Tensor`.
- **coeff** (*int*) – Coefficient value. - `coeff < 1`, the image is darkened. - `coeff = 1`, the image is unchanged. - `coeff > 1`, the image is lightened.
- **rgb** (*bool*) – Operate on RGB images. Default: True.

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.sunrise(rgb=True)
>> filtered = caer.transforms.adjust_brightness(tens, coeff=1.4, rgb=True)
>> filtered
(427, 640, 3)
```

caer.transforms.**adjust_contrast**(*tens*, *contrast_factor*)

Adjust contrast of an image.

**Parameters**

- **tens** (*Tensor*) – Any valid `caer.Tensor`.
- **contrast_factor** (*float*) – How much to adjust the contrast. Can be any non negative number. 0 gives a solid gray image, 1 gives the original image while 2 increases the contrast by a factor of 2.

**Returns** Contrast adjusted image.

**Return type** `caer.Tensor`

caer.transforms.**adjust_gamma**(*tens*, *gamma*, *gain=1*)

Perform gamma correction on an image.

Also known as Power Law Transform. Intensities in RGB mode are adjusted based on the following equation: .. math:

```
I_{\text{out}} = 255 \times \text{gain} \times \left(\frac{I_{\text{in}}}
↪{255}\right)^{\gamma}
```

See `Gamma Correction`_ for more details. .. _Gamma Correction: https://en.wikipedia.org/wiki/Gamma_correction

**Parameters**

- **tens** (*Tensor*) – Any valid `caer.Tensor`.
- **gamma** (*float*) – Non negative real number, same as $\gamma$ in the equation. gamma larger than 1 make the shadows darker, while gamma smaller than 1 make dark regions lighter.
- **gain** (*float*) – The constant multiplier.

Examples:

```
>> tens = caer.data.sunrise(rgb=True)
>> filtered = caer.transforms.adjust_gamma(tens, gamma=1.5, rgb=True)
>> filtered
(427, 640, 3)
```

**Return type** Tensor

`caer.transforms.`**adjust_hue**(*tens*, *hue_factor*)

Adjust hue of an image.

The image hue is adjusted by converting the image to HSV and cyclically shifting the intensities in the hue channel (H). The image is then converted back to original image mode.

See `Hue`_ for more details. .. _Hue: https://en.wikipedia.org/wiki/Hue

**Parameters**

- **tens** (*Tensor*) – Any valid `caer.Tensor`.
- **hue_factor** (*float*) – How much to shift the hue channel. Should be in the range [-0.5, 0.5]. 0.5 and -0.5 give complete reversal of hue channel in HSV space in positive and negative direction respectively. 0 means no shift. Therefore, both -0.5 and 0.5 will give an image with complementary colors while 0 gives the original image.

**Return type** Tensor

**Returns** Hue adjusted image.

Examples:

```
>> tens = caer.data.sunrise(rgb=True)
>> filtered = caer.transforms.adjust_hue(tens, hue_factor=1, rgb=True)
>> filtered
(427, 640, 3)
```

`caer.transforms.`**adjust_saturation**(*tens*, *saturation_factor*)

Adjust color saturation of an image.

**Parameters**

- **tens** (*Tensor*) – Any valid `caer.Tensor`.

- **saturation_factor** (*float*) – How much to adjust the saturation. 0 will give a black and white image, 1 will give the original image while 2 will enhance the saturation by a factor of 2.

**Return type** Tensor

**Returns** Saturation-adjusted image.

Examples:

```
>> tens = caer.data.sunrise(rgb=True)
>> filtered = caer.transforms.adjust_saturation(tens, saturation_factor=1.5,␣
↪rgb=True)
>> filtered
(427, 640, 3)
```

caer.transforms.**affine**(*tens*, *angle*, *translate*, *scale*, *shear*, *interpolation='bilinear'*, *mode=0*, *fillcolor=0*)

Apply affine transformation on the image keeping image center invariant.

**Parameters**

- **tens** (*Tensor*) – Any valid caer.Tensor.
- **angle** (*float or int*) – Rotation angle in degrees between -180 and 180, clockwise direction.
- **translate** (*list or tuple of integers*) – Horizontal and vertical translations (post-rotation translation)
- **scale** (*float*) – Overall scale
- **shear** (*float*) – Shear angle value in degrees between -180 to 180, clockwise direction.
- **interpolation** (*int, str*) – Interpolation to use for resizing. Defaults to *'bilinear'*. Supports *'bilinear'*, *'bicubic'*, *'area'*, *'nearest'*.
- **mode** (*int, str*) – Method for filling in border regions. Defaults to constant meaning areas outside the image are filled with a value (val, default 0). Supports 'replicate', 'reflect', 'reflect-101'.
- **val** (*int*) – Optional fill color for the area outside the transform in the output image. Default: 0

**Return type** Tensor

caer.transforms.**brighten**(*tens*, *coeff=- 1*, *rgb=True*)

Brighten an image.

**Parameters**

- **tens** (*Tensor*) – Any regular caer.Tensor.
- **coeff** (*int*) – Coefficient value.
- **rgb** (*bool*) – Operate on RGB images. Default: True.

**Return type** Tensor

**Returns** Tensor of shape (height, width, channels).

Examples:

```
>> tens = caer.data.sunrise(rgb=True)
>> filtered = caer.transforms.brighten(tens, coeff=-1, rgb=True)
>> filtered
(427, 640, 3)
```

caer.transforms.**correct_exposure**(*tens*, *rgb=True*)

Correct the exposure of an image.

**Parameters**

- **tens** (*Tensor*) – Any regular `caer.Tensor`.
- **rgb** (*bool*) – Operate on RGB images. Default: True.

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.sunrise(rgb=True)
>> filtered = caer.transforms.correct_exposure(tens, rgb=True)
>> filtered
(427, 640, 3)
```

caer.transforms.**darken**(*tens*, *darkness_coeff=- 1*)

Darken an image.

**Parameters**

- **tens** (*Tensor*) – Any regular `caer.Tensor`.
- **darkness_coeff** (*int*) – Coefficient value.
- **rgb** (*bool*) – Operate on RGB images. Default: True.

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.sunrise(rgb=True)
>> filtered = caer.transforms.darken(tens, coeff=-1, rgb=True)
>> filtered
(427, 640, 3)
```

caer.transforms.**equalize**(*tens*, *mask=None*, *by_channels=True*)

Equalize the image histogram.

**Parameters**

- **tens** (*Tensor*) – RGB or grayscale image.
- **mask** (*Tensor*) – An optional mask. If given, only the pixels selected by the mask are included in the analysis. Maybe 1 channel or 3 channel array.
- **by_channels** (*bool*) – If True, use equalization by channels separately, else convert image to YCbCr representation and use equalization by *Y* channel.

**Return type** Tensor

**Returns** Equalized image (Tensor)

Examples:

```
>> tens = caer.data.beverages()
>> equalized = caer.equalize(tens, mask=None)
>> equalized.shape
(427,640,3)
```

caer.transforms.**hflip**(*tens*)

Flip an image horizontally.

**Parameters** **tens** (*Tensor*) – Image to be flipped.

**Return type** Tensor

**Returns** Flipped image.

caer.transforms.**hvflip**(*tens*)

Flip an image both horizontally and vertically.

**Parameters** **tens** (*Tensor*) – Image to be flipped.

**Return type** Tensor

**Returns** Flipped image.

caer.transforms.**pad**(*tens*, *padding*, *fill=0*, *padding_mode='constant'*)

Pad the given image on all sides with specified padding mode and fill value.

**Parameters**

- **tens** (*Tensor*) – image to be padded.
- **padding** (*int or tuple*) – Padding on each border. If a single int is provided this is used to pad all borders. If tuple of length 2 is provided this is the padding on left/right and top/bottom respectively. If a tuple of length 4 is provided this is the padding for the left, top, right and bottom borders respectively.
- **fill** – Pixel fill value for constant fill. Default is 0. If a tuple of length 3, it is used to fill R, G, B channels respectively. This value is only used when the padding_mode is constant
- **padding_mode** – Type of padding. Should be: constant, edge, reflect or symmetric. Default is constant. - constant: pads with a constant value, this value is specified with fill - edge: pads with the last value on the edge of the image - reflect: pads with reflection of image (without repeating the last value on the edge)

  padding [1, 2, 3, 4] with 2 elements on both sides in reflect mode will result in [3, 2, 1, 2, 3, 4, 3, 2]

  – **symmetric: pads with reflection of image (repeating the last value on the edge)**
  padding [1, 2, 3, 4] with 2 elements on both sides in symmetric mode will result in [2, 1, 1, 2, 3, 4, 4, 3]

**Return type** Tensor

**Returns** Tensor of shape `(height, width, channels)`.

caer.transforms.**posterize**(*tens*, *bits*)

Reduce the number of bits for each color channel in the image.

**Parameters**

- **tens** (*Tensor*) – Image to posterize.
- **bits** (*int*) – Number of high bits. Must be in range [0, 8]

**Return type** `Tensor`

**Returns** Image with reduced color channels (Tensor)

Examples:

```
>> tens = caer.data.sunrise()
>> posterized = caer.posterize(tens, bits=4)
>> posterized.shape
(427,640,3)
```

caer.transforms.**rand_flip**(*tens*)

Randomly flip an image vertically or horizontally.

**Parameters** **tens** (*Tensor*) – Image to be flipped.

**Return type** `Tensor`

**Returns** Flipped image.

caer.transforms.**random_brightness**(*tens*)

Add random brightness to an image.

**Parameters** **tens** (*Tensor*) – Any regular `caer.Tensor`.

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.sunrise()
>> filtered = caer.transforms.random_brightness(tens, rgb=True)
>> filtered
(427, 640, 3)
```

caer.transforms.**rotate**(*tens*, *angle*, *rotPoint=None*)

Rotates an given image by an angle around a particular rotation point (if provided) or centre otherwise.

**Return type** `Tensor`

caer.transforms.**sim_autumn**(*tens*)

Simulate autumn conditions on an image.

**Parameters** **tens** (*Tensor*) – Any regular `caer.Tensor`.

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.sunrise
>> filtered = caer.filters.sim_autumn(tens)
>> filtered
(427, 640, 3)
```

caer.transforms.**sim_fog**(*tens*, *fog_coeff=- 1*)

Simulate foggy conditions on an image.

**Parameters**

- **tens** (*Tensor*) – Any regular `caer.Tensor`.
- **fog_coeff** (*int*) – Coefficient value.

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.sunrise
>> filtered = caer.filters.sim_fog(tens, fog_coeff=-1)
>> filtered
(427, 640, 3)
```

caer.transforms.**sim_gravel**(*tens*, *rectangular_roi=(- 1, - 1, - 1, - 1)*, *num_patches=8*)

Simulate gravelly conditions on an image.

**Parameters**

- **tens** (*Tensor*) – Any regular `caer.Tensor`.
- **rectangular_roi** (*tuple*) – Rectanglar co-ordinates of the intended region of interest. Default: (-1,-1,-1,-1).
- **num_patches** (*int*) – Number of patches to operate on.

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.sunrise
>> filtered = caer.filters.sim_gravel(tens)
>> filtered
(427, 640, 3)
```

caer.transforms.**sim_motion_blur**(*tens*, *speed_coeff=- 1*)

Simulate motion-blur conditions on an image.

**Parameters**

- **tens** (*Tensor*) – Any regular `caer.Tensor`.
- **speed_coeff** (*int, float*) – Speed coefficient. Value must be between 0 and 1.

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.sunrise
>> filtered = caer.filters.sim_motion_blur(tens, speed_coeff=-1)
>> filtered
(427, 640, 3)
```

`caer.transforms.`**sim_rain**(*tens*, *slant=- 1*, *drop_length=20*, *drop_width=1*, *drop_color=(200, 200, 200)*, *rain_type='None'*)

Simulate rainy conditions on an image.

**Parameters**

- **tens** (*Tensor*) – Any regular `caer.Tensor`.
- **slant** (*int*) – Slant value.
- **drop_length** (*int*) – Length of the raindrop.
- **drop_width** (*int*) – Width of the raindrop.
- **drop_color** (*tuple*) – Color of the raindrop.
- **rain_type** (*str*) – Type of rain. Can be either 'drizzle', 'heavy' or 'torrential'.

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.sunrise
>> filtered = caer.filters.sim_rain(tens)
>> filtered
(427, 640, 3)
```

`caer.transforms.`**sim_shadow**(*tens*, *num_shadows=1*, *rectangular_roi=(- 1, - 1, - 1, - 1)*, *shadow_dimension=5*)

Simulate shadowy conditions on an image.

**Parameters**

- **tens** (*Tensor*) – Any regular `caer.Tensor`.
- **num_shadows** (*int*) – Number of shadows to work with. Value must be between 1 and 10.
- **rectangular_roi** (*tuple*) – Rectanglar co-ordinates of the intended region of interest. Default: (-1,-1,-1,-1).
- **shadow_dimensions** (*int*) – Number of shadow dimensions. Value must be > 3.

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.sunrise
>> filtered = caer.filters.sim_shadow(tens)
>> filtered
(427, 640, 3)
```

caer.transforms.**sim_snow**(*tens*, *snow_coeff=- 1*)

Simulate snowy conditions on an image.

**Parameters**

- **tens** (*Tensor*) – Any regular `caer.Tensor`.
- **snow_coeff** (*int*) – Coefficient value.

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.sunrise
>> filtered = caer.filters.sim_snow(tens, snow_coeff=-1)
>> filtered
(427, 640, 3)
```

caer.transforms.**sim_sun_flare**(*tens*, *flare_center=- 1*, *angle=- 1*, *num_flare_circles=8*, *src_radius=400*, *src_color=(255, 255, 255)*)

Add a source of light (flare) on an specific region of an image.

**Parameters**

- **tens** (*Tensor*) – Any regular `caer.Tensor`.
- **flare_center** (*int*) – Center of the flare. Default: -1.
- **angle** (*int*) – Angle of the flare. Default: -1
- **num_flare_circles** (*int*) – Number of flare circles to operate on.
- **src_radius** (*int*) – Intended size of the flare
- **src_color** (*tuple*) – Intended source color of the flare

**Return type** `Tensor`

**Returns** Tensor of shape `(height, width, channels)`.

Examples:

```
>> tens = caer.data.sunrise
>> filtered = caer.filters.sim_sun_flare(tens)
>> filtered
(427, 640, 3)
```

caer.transforms.**solarize**(*tens*, *threshold=128*)

Invert all pixel values above a threshold.

**Parameters**

- **tens** (*Tensor*) – The image to solarize.
- **threshold** (*int*) – All pixels above this grayscale level are inverted.

**Return type** Tensor

**Returns** Solarized image (Tensor)

Examples:

```
>> tens = caer.data.sunrise()
>> solarized = caer.solarize(tens, threshold=128)
>> solarized.shape
(427,640,3)
```

caer.transforms.**translate**(*image*, *x*, *y*)

Translates a given image across the x-axis and the y-axis

**Parameters**

- **x** (*int*) – shifts the image right (positive) or left (negative)
- **y** (*int*) – shifts the image down (positive) or up (negative)

**Return type** Tensor

**Returns** The translated image

caer.transforms.**vflip**(*tens*)

Flip an image vertically.

**Parameters** **tens** (*Tensor*) – Image to be flipped.

**Return type** Tensor

**Returns** Flipped image.

CHAPTER

# TEN

# CAER.VIDEO

## 10.1 Video Streams

### 10.1.1 Stream

caer.video.**Stream**(*source=0*, *qsize=128*)

This is an auxiliary class that enables Video Streaming for `caer` with minimalistic latency, and at the expense of little to no additional computational requirements.

The basic idea behind it is to tracks and save the salient feature array for the given number of frames and then uses these anchor point to cancel out all perturbations relative to it for the incoming frames in the queue. This class relies heavily on **Threaded Queue mode** for error-free & ultra-fast frame handling.

**Parameters**

- **source** (*int, str*) – Source path for the video. Uses an external camera device if `source` is an integer.
- **qsize** (*int*) – Default queue size for handling the video streams. Default: 128.

## 10.2 LiveStream

caer.video.**LiveStream**(*source=0*)

This is an auxiliary class that enables Live Video Streaming for caer with minimalistic latency, and at the expense of little to no additional computational requirements.

The basic idea behind it is to tracks and save the salient feature array for the given number of frames and then uses these anchor point to cancel out all perturbations relative to it for the incoming frames in the queue. This class relies heavily on **Threaded Queue mode** for error-free & ultra-fast frame handling.

**Parameters** **source** (*int*) – Source path for the video. If `source=0`, the default camera device is used. For multiple external camera devices, use incremented values. For eg: `source=1` represents the second camera device on your system.

---

**Frame Extraction**

### 10.2.1 extract_frames

caer.video.**extract_frames**(*input_folder*, *output_folder*, *target_size=None*, *recursive=False*, *label_counter=None*, *max_video_count=None*, *frames_per_sec=None*, *frame_interval=None*, *dest_filetype='jpg'*)

Extract frames from videos within a directory and save them as separate frames in an output directory.

**Parameters**

- **input_folder** (*str*) – Input video directory.
- **output_folder** (*str*) – Output directory to save the frames.
- **target_size** (*tuple*) – Destination Image Size (tuple of size 2)
- **label_counter** (*int*) – Starting label counter (optional)
- **max_video_count** (*int*) – Number of videos to process.
- **frames_per_sec** (*int, float*) – Number of frames to process per second.
- **frame_interval** (*int, float*) – Interval between the frames to be processed.
- **dest_filetype** (*str*) – Processed image filetype (png, jpg). Default: png

**Return type** int

**Returns** label_counter (after processing)

---

**Video Stablizer**

### 10.2.2 Stablizer

CHAPTER

ELEVEN

# CONTRIBUTOR CODE OF CONDUCT

## 11.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to make participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

## 11.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

## 11.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

## 11.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

## 11.5 Enforcement

Instances of abusive, harassing or otherwise unacceptable behavior may be reported by contacting the project team at jasmcaus@gmail.com. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

## 11.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available here.

CHAPTER

# TWELVE

# CAER GOVERNANCE | PERSONS OF INTEREST

## 12.1 Leads

- Jason Dsouza (jasmcaus) (Caer founder)

## 12.2 Core Maintainers

- Jason Dsouza (jasmcaus) (Caer founder)

CHAPTER

THIRTEEN

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## C

# INDEX