# Blend Modes Documentation

***Release 2.0.1***

**Florian Roscheck**

**Jul 17, 2019**

# Contents

This Python package implements blend modes for images.

CHAPTER 1

# Description

The Blend Modes package enables blending different images, or image layers, by means of blend modes. These modes are commonly found in graphics programs like Adobe Photoshop or GIMP.

Blending through blend modes allows to mix images in a variety of ways. This package currently supports the following blend modes (name of the respective functions in the package in `italics`):

- Soft Light (`blend_modes.soft_light`)
- Lighten Only (`blend_modes.lighten_only`)
- Dodge (`blend_modes.dodge`)
- Addition (`blend_modes.addition`)
- Darken Only (`blend_modes.darken_only`)
- Multiply (`blend_modes.multiply`)
- Hard Light (`blend_modes.hard_light`)
- Difference (`blend_modes.difference`)
- Subtract (`blend_modes.subtract`)
- Grain Extract (known from GIMP, `blend_modes.grain_extract`)
- Grain Merge (known from GIMP, `blend_modes.grain_merge`)
- Divide (`blend_modes.divide`)
- Overlay (`blend_modes.overlay`)
- Normal (`blend_modes.normal`)

The intensity of blending can be controlled by means of an *opacity* parameter that is passed into the functions. See *Usage* for more information.

The Blend Modes package is optimized for speed. It takes advantage of vectorization through Numpy. Further speedup can be achieved when implementing the package in Cython. However, Cython implementation is not part of this package.

CHAPTER 2

# Usage

The blend mode functions take image data expressed as arrays as an input. These image data are usually obtained through functions from image processing packages. Two popular image processing packages in Python are PIL or its fork Pillow and OpenCV. The examples in this chapter show how to blend images using these packages.

## 2.1 Input and Output Formats

A typical blend mode operation is called like this:

```
blended_img = soft_light(bg_img, fg_img, opacity)
```

The blend mode functions expect Numpy float arrays in the format [*pixels in dimension 1*,*pixels in dimension 2*,4] as an input. Both images needs to have the same size, so the *pixels in dimension 1* must be the same for `bg_img` and `fg_img`. Same applies to the *pixels in dimension 2*. Thus, a valid shape of the arrays would be `bg_img.shape == (640,320,4)` and `fg_img.shape == (640,320,4)`.

The order of the channels in the third dimension should be *R, G, B, A*, where *A* is the alpha channel. All values should be *floats* in the range *0.0 <= value <= 255.0*.

The blend mode functions return arrays in the same format as the input format.

## 2.2 Examples

The following examples show how to use the Blend Modes package in typical applications.

The examples are structured in three parts:

1. Load background and foreground image. The foreground image is to be blended onto the background image.
2. Use the Blend Modes package to blend the two images via the "soft light" blend mode. The package supports multiple blend modes. See the *Description* for a full list.
3. Display the blended image.

### 2.2.1 PIL/Pillow Example

The following example shows how to use the Blend Modes package with the PIL or Pillow packages.

```
from PIL import Image
import numpy
from blend_modes import soft_light

# Import background image
background_img_raw = Image.open('background.png')  # RGBA image
background_img = numpy.array(background_img_raw)  # Inputs to blend_modes need to be␣
↪numpy arrays.
background_img_float = background_img.astype(float)  # Inputs to blend_modes need to␣
↪be floats.

# Import foreground image
foreground_img_raw = Image.open('foreground.png')  # RGBA image
foreground_img = numpy.array(foreground_img_raw)  # Inputs to blend_modes need to be␣
↪numpy arrays.
foreground_img_float = foreground_img.astype(float)  # Inputs to blend_modes need to␣
↪be floats.

# Blend images
opacity = 0.7  # The opacity of the foreground that is blended onto the background is␣
↪70 %.
blended_img_float = soft_light(background_img_float, foreground_img_float, opacity)

# Convert blended image back into PIL image
blended_img = numpy.uint8(blended_img_float)  # Image needs to be converted back to␣
↪uint8 type for PIL handling.
blended_img_raw = Image.fromarray(blended_img)  # Note that alpha channels are␣
↪displayed in black by PIL by default.
                                                # This behavior is difficult to␣
↪change (although possible).
                                                # If you have alpha channels in your␣
↪images, then you should give
                                                # OpenCV a try.

# Display blended image
blended_img_raw.show()
```

### 2.2.2 OpenCV Example

The following example shows how to use the Blend Modes package with OpenCV.

```
import cv2  # import OpenCV
import numpy
from blend_modes import soft_light

# Import background image
background_img_float = cv2.imread('background.png',-1).astype(float)

# Import foreground image
foreground_img_float = cv2.imread('foreground.png',-1).astype(float)

# Blend images
```

(continues on next page)

(continued from previous page)

```
opacity = 0.7  # The opacity of the foreground that is blended onto the background is␣
↪70 %.
blended_img_float = soft_light(background_img_float, foreground_img_float, opacity)

# Display blended image
blended_img_uint8 = blended_img_float.astype(numpy.uint8)  # Convert image to OpenCV␣
↪native display format
cv2.imshow('window', blended_img_uint8)
cv2.waitKey()  # Press a key to close window with the image.
```

CHAPTER 3

# Installation

The Blend Modes package can be installed through pip:

```
pip install blend_modes
```

CHAPTER 4

# Dependencies

The Blend Modes package needs Numpy to function correctly. For loading images the following packages have been successfully used:

- PIL
- Pillow
- OpenCV

CHAPTER 5

# See Also

Blend modes are further described on Wikipedia. An actual implementation can be found in the GIMP source code.

CHAPTER 6

# Contribution

I am happy about any contribution or feedback. Please let me know about your comments via the Issues tab on GitHub.

CHAPTER 7

# License

The Blend Modes package is distributed under the MIT License (MIT). Please also take note of the licenses of the dependencies.

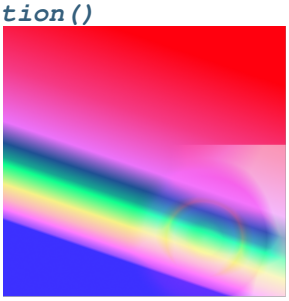
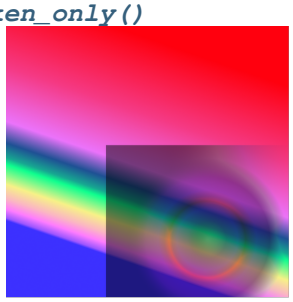
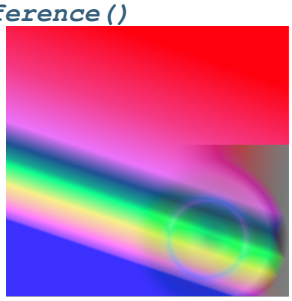
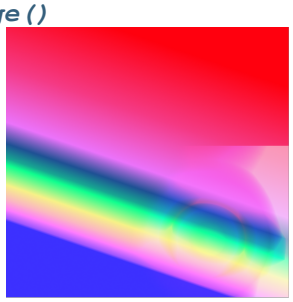
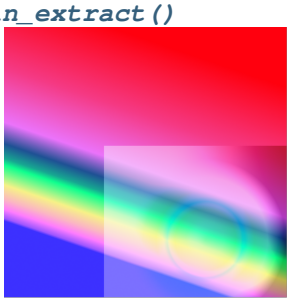
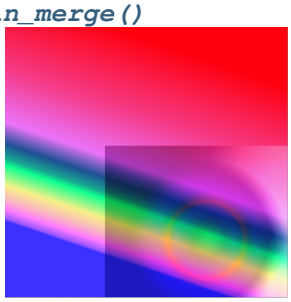
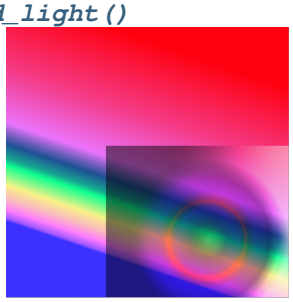
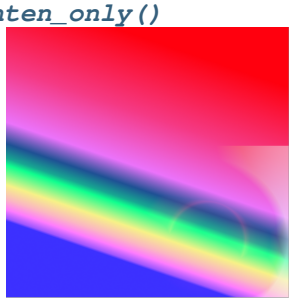
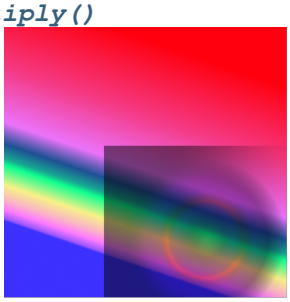
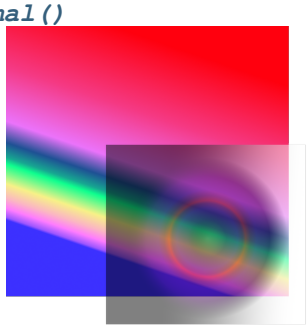
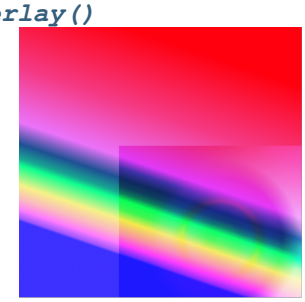
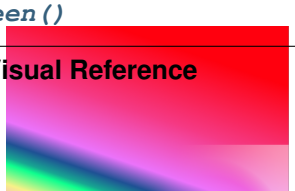
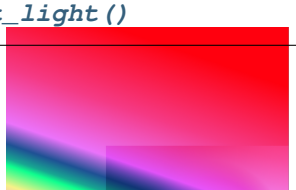
CHAPTER 8

Contents

## 8.1 Visual Reference

This page serves as a visual reference for the blend modes available in this Python package. Click on the title on top of an image to get to the documentation of the related blending function.

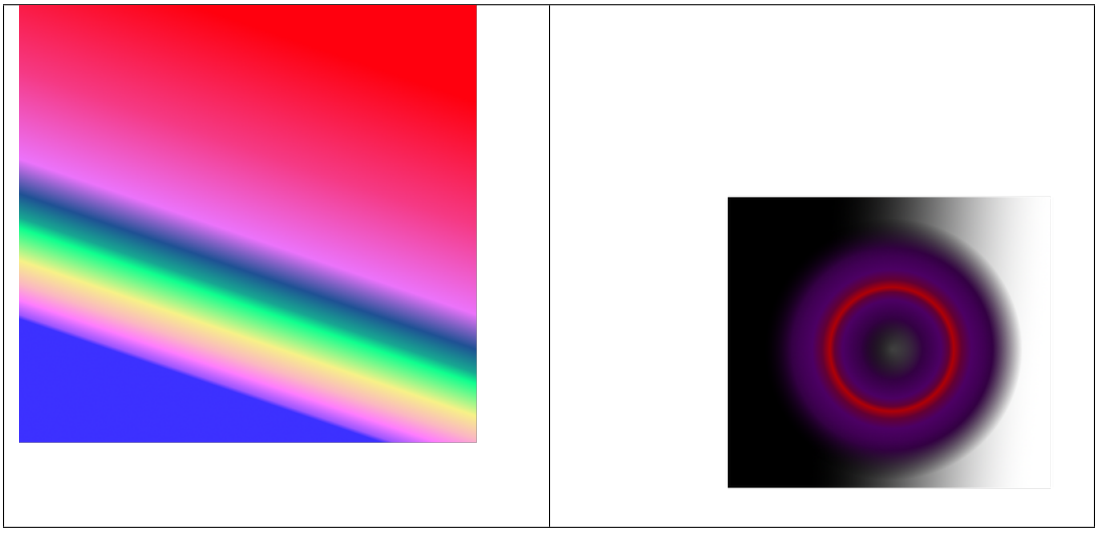
### 8.1.1 Panel of Examples

| | | |
|---|---|---|
| *addition()*  | *darken_only()*  | *difference()*  |
| *divide()*  | *dodge()*  | *grain_extract()*  |
| *grain_merge()*  | *hard_light()*  | *lighten_only()*  |
| *multiply()*  | *normal()*  | *overlay()*  |
| *screen()*  | *soft_light()*  | *subtract()*  |

### 8.1.2 About the Blended Images

All examples are blends of two images: As a bottom layer, there is a rainbow-filled square with some transparent border on the right and bottom edges. The top layer is a small rectangle that is filled with a colorful circular gradient. The top layer is blended upon the bottom layer with 50% transparency in all of the images in the Panel of Examples above.

Table 1: Bottom and top layers for blending examples



## 8.2 Reference

This page documents all function available in blend_modes in detail. If this documentation cannot answer your questions, please raise an issue on blend_modes' GitHub page.

### 8.2.1 Overview

| | |
|---|---|
| *addition* | Apply addition blending mode of a layer on an image. |
| *darken_only* | Apply darken only blending mode of a layer on an image. |
| *difference* | Apply difference blending mode of a layer on an image. |
| *divide* | Apply divide blending mode of a layer on an image. |
| *dodge* | Apply dodge blending mode of a layer on an image. |
| *grain_extract* | Apply grain extract blending mode of a layer on an image. |
| *grain_merge* | Apply grain merge blending mode of a layer on an image. |
| *hard_light* | Apply hard light blending mode of a layer on an image. |
| *lighten_only* | Apply lighten only blending mode of a layer on an image. |
| *multiply* | Apply multiply blending mode of a layer on an image. |
| *normal* | Apply "normal" blending mode of a layer on an image. |
| *overlay* | Apply overlay blending mode of a layer on an image. |

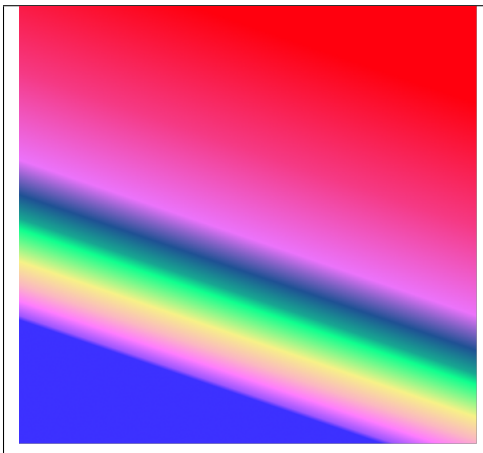
Continued on next page

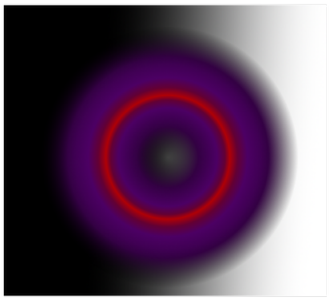
Table 2 – continued from previous page

| | |
|---|---|
| *screen* | Apply screen blending mode of a layer on an image. |
| *soft_light* | Apply soft light blending mode of a layer on an image. |
| *subtract* | Apply subtract blending mode of a layer on an image. |

**Note:** All examples on this page are blends of two images: As a bottom layer, there is a rainbow-filled square with some transparent border on the right and bottom edges. The top layer is a small rectangle that is filled with a colorful circular gradient. The top layer is blended upon the bottom layer with 50% transparency in all of the examples below.

Table 3: Bottom and top layers for blending examples



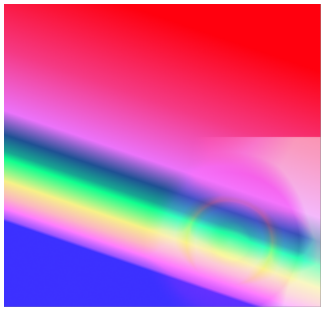


### 8.2.2 Detailed Documentation

**addition**(*img_in*, *img_layer*, *opacity*, *disable_type_checks: bool = False*)

Apply addition blending mode of a layer on an image.

#### Example



```
import cv2, numpy
from blend_modes import addition
img_in = cv2.imread('./orig.png', -1).astype(float)
img_layer = cv2.imread('./layer.png', -1).astype(float)
img_out = addition(img_in,img_layer,0.5)
cv2.imshow('window', img_out.astype(numpy.uint8))
cv2.waitKey()
```

**See also:**

Find more information on Wikipedia.

**Parameters**

- **img_in** (*3-dimensional numpy array of floats (r/g/b/a) in range 0-255.0*) – Image to be blended upon
- **img_layer** (*3-dimensional numpy array of floats (r/g/b/a) in range 0.0-255.0*) – Layer to be blended with image
- **opacity** (*float*) – Desired opacity of layer for blending
- **disable_type_checks** (*bool*) – Whether type checks within the function should be disabled. Disabling the checks may yield a slight performance improvement, but comes at the cost of user experience. If you are certain that you are passing in the right arguments, you may set this argument to 'True'. Defaults to 'False'.

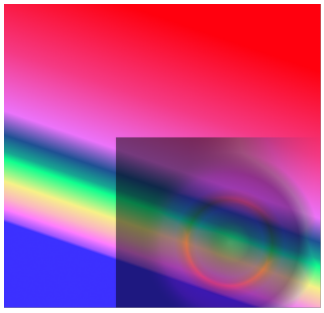
**Returns** Blended image

**Return type** 3-dimensional numpy array of floats (r/g/b/a) in range 0.0-255.0

**darken_only**(*img_in*, *img_layer*, *opacity*, *disable_type_checks: bool = False*)

Apply darken only blending mode of a layer on an image.

### Example



```
import cv2, numpy
from blend_modes import darken_only
img_in = cv2.imread('./orig.png', -1).astype(float)
img_layer = cv2.imread('./layer.png', -1).astype(float)
img_out = darken_only(img_in,img_layer,0.5)
cv2.imshow('window', img_out.astype(numpy.uint8))
cv2.waitKey()
```

**See also:**

Find more information on Wikipedia.

**Parameters**

- **img_in** (*3-dimensional numpy array of floats (r/g/b/a) in range 0-255.0*) – Image to be blended upon
- **img_layer** (*3-dimensional numpy array of floats (r/g/b/a) in range 0.0-255.0*) – Layer to be blended with image
- **opacity** (*float*) – Desired opacity of layer for blending
- **disable_type_checks** (*bool*) – Whether type checks within the function should be disabled. Disabling the checks may yield a slight performance improvement, but comes at the cost of user experience. If you are certain that you are passing in the right arguments, you may set this argument to 'True'. Defaults to 'False'.

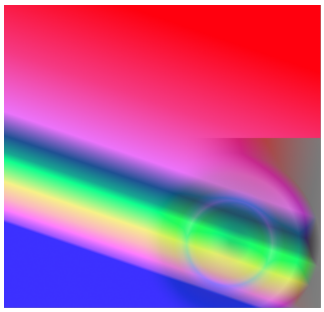
**Returns** Blended image

**Return type** 3-dimensional numpy array of floats (r/g/b/a) in range 0.0-255.0

**difference**(*img_in*, *img_layer*, *opacity*, *disable_type_checks: bool = False*)

Apply difference blending mode of a layer on an image.

### Example



```
import cv2, numpy
from blend_modes import difference
img_in = cv2.imread('./orig.png', -1).astype(float)
img_layer = cv2.imread('./layer.png', -1).astype(float)
img_out = difference(img_in,img_layer,0.5)
cv2.imshow('window', img_out.astype(numpy.uint8))
cv2.waitKey()
```

**See also:**

Find more information on Wikipedia.

**Parameters**

- **img_in** (*3-dimensional numpy array of floats (r/g/b/a) in range 0-255.0*) – Image to be blended upon
- **img_layer** (*3-dimensional numpy array of floats (r/g/b/a) in range 0.0-255.0*) – Layer to be blended with image

- **opacity** (*float*) – Desired opacity of layer for blending
- **disable_type_checks** (*bool*) – Whether type checks within the function should be disabled. Disabling the checks may yield a slight performance improvement, but comes at the cost of user experience. If you are certain that you are passing in the right arguments, you may set this argument to 'True'. Defaults to 'False'.

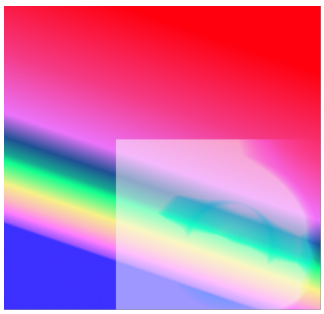
**Returns** Blended image

**Return type** 3-dimensional numpy array of floats (r/g/b/a) in range 0.0-255.0

**divide**(*img_in*, *img_layer*, *opacity*, *disable_type_checks: bool = False*)

Apply divide blending mode of a layer on an image.

### Example



```
import cv2, numpy
from blend_modes import divide
img_in = cv2.imread('./orig.png', -1).astype(float)
img_layer = cv2.imread('./layer.png', -1).astype(float)
img_out = divide(img_in,img_layer,0.5)
cv2.imshow('window', img_out.astype(numpy.uint8))
cv2.waitKey()
```

**See also:**

Find more information on Wikipedia.

**Parameters**

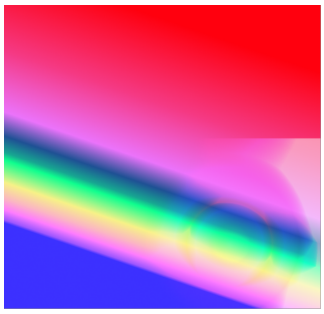
- **img_in** (*3-dimensional numpy array of floats (r/g/b/a) in range 0-255.0*) – Image to be blended upon
- **img_layer** (*3-dimensional numpy array of floats (r/g/b/a) in range 0.0-255.0*) – Layer to be blended with image
- **opacity** (*float*) – Desired opacity of layer for blending
- **disable_type_checks** (*bool*) – Whether type checks within the function should be disabled. Disabling the checks may yield a slight performance improvement, but comes at the cost of user experience. If you are certain that you are passing in the right arguments, you may set this argument to 'True'. Defaults to 'False'.

**Returns** Blended image

**Return type** 3-dimensional numpy array of floats (r/g/b/a) in range 0.0-255.0

**dodge** (*img_in*, *img_layer*, *opacity*, *disable_type_checks: bool = False*)
Apply dodge blending mode of a layer on an image.

### Example



```
import cv2, numpy
from blend_modes import dodge
img_in = cv2.imread('./orig.png', -1).astype(float)
img_layer = cv2.imread('./layer.png', -1).astype(float)
img_out = dodge(img_in,img_layer,0.5)
cv2.imshow('window', img_out.astype(numpy.uint8))
cv2.waitKey()
```

**See also:**

Find more information on Wikipedia.

**Parameters**

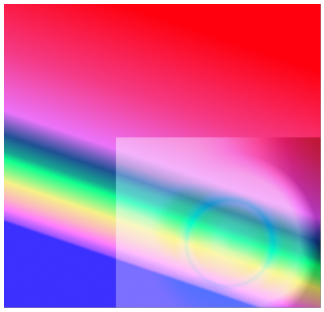
- **img_in** (*3-dimensional numpy array of floats (r/g/b/a) in range 0-255.0*) – Image to be blended upon
- **img_layer** (*3-dimensional numpy array of floats (r/g/b/a) in range 0.0-255.0*) – Layer to be blended with image
- **opacity** (*float*) – Desired opacity of layer for blending
- **disable_type_checks** (*bool*) – Whether type checks within the function should be disabled. Disabling the checks may yield a slight performance improvement, but comes at the cost of user experience. If you are certain that you are passing in the right arguments, you may set this argument to 'True'. Defaults to 'False'.

**Returns** Blended image

**Return type** 3-dimensional numpy array of floats (r/g/b/a) in range 0.0-255.0

**grain_extract** (*img_in*, *img_layer*, *opacity*, *disable_type_checks: bool = False*)
Apply grain extract blending mode of a layer on an image.

### Example



```
import cv2, numpy
from blend_modes import grain_extract
img_in = cv2.imread('./orig.png', -1).astype(float)
img_layer = cv2.imread('./layer.png', -1).astype(float)
img_out = grain_extract(img_in,img_layer,0.5)
cv2.imshow('window', img_out.astype(numpy.uint8))
cv2.waitKey()
```

**See also:**

Find more information in the GIMP Documentation.

**Parameters**

- **img_in** (*3-dimensional numpy array of floats (r/g/b/a) in range 0-255.0*) – Image to be blended upon
- **img_layer** (*3-dimensional numpy array of floats (r/g/b/a) in range 0.0-255.0*) – Layer to be blended with image
- **opacity** (*float*) – Desired opacity of layer for blending
- **disable_type_checks** (*bool*) – Whether type checks within the function should be disabled. Disabling the checks may yield a slight performance improvement, but comes at the cost of user experience. If you are certain that you are passing in the right arguments, you may set this argument to 'True'. Defaults to 'False'.

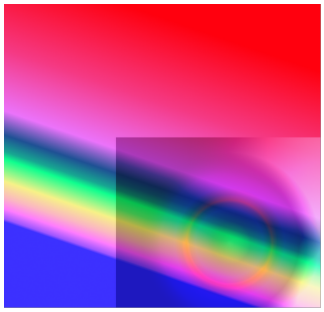
**Returns** Blended image

**Return type** 3-dimensional numpy array of floats (r/g/b/a) in range 0.0-255.0

**grain_merge**(*img_in*, *img_layer*, *opacity*, *disable_type_checks: bool = False*)

Apply grain merge blending mode of a layer on an image.

### Example



```
import cv2, numpy
from blend_modes import grain_merge
img_in = cv2.imread('./orig.png', -1).astype(float)
img_layer = cv2.imread('./layer.png', -1).astype(float)
img_out = grain_merge(img_in,img_layer,0.5)
cv2.imshow('window', img_out.astype(numpy.uint8))
cv2.waitKey()
```

**See also:**

Find more information in the GIMP Documentation.

**Parameters**

- **img_in** (*3-dimensional numpy array of floats (r/g/b/a) in range 0-255.0*) – Image to be blended upon
- **img_layer** (*3-dimensional numpy array of floats (r/g/b/a) in range 0.0-255.0*) – Layer to be blended with image
- **opacity** (*float*) – Desired opacity of layer for blending
- **disable_type_checks** (*bool*) – Whether type checks within the function should be disabled. Disabling the checks may yield a slight performance improvement, but comes at the cost of user experience. If you are certain that you are passing in the right arguments, you may set this argument to 'True'. Defaults to 'False'.

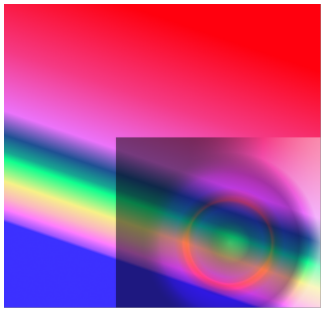
**Returns** Blended image

**Return type** 3-dimensional numpy array of floats (r/g/b/a) in range 0.0-255.0

**hard_light**(*img_in*, *img_layer*, *opacity*, *disable_type_checks: bool = False*)

Apply hard light blending mode of a layer on an image.

### Example



```
import cv2, numpy
from blend_modes import hard_light
img_in = cv2.imread('./orig.png', -1).astype(float)
img_layer = cv2.imread('./layer.png', -1).astype(float)
img_out = hard_light(img_in,img_layer,0.5)
cv2.imshow('window', img_out.astype(numpy.uint8))
cv2.waitKey()
```

**See also:**

Find more information on Wikipedia.

**Parameters**

- **img_in** (*3-dimensional numpy array of floats (r/g/b/a) in range 0-255.0*) – Image to be blended upon
- **img_layer** (*3-dimensional numpy array of floats (r/g/b/a) in range 0.0-255.0*) – Layer to be blended with image
- **opacity** (*float*) – Desired opacity of layer for blending
- **disable_type_checks** (*bool*) – Whether type checks within the function should be disabled. Disabling the checks may yield a slight performance improvement, but comes at the cost of user experience. If you are certain that you are passing in the right arguments, you may set this argument to 'True'. Defaults to 'False'.

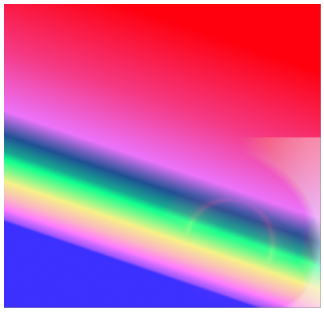
**Returns** Blended image

**Return type** 3-dimensional numpy array of floats (r/g/b/a) in range 0.0-255.0

**lighten_only**(*img_in*, *img_layer*, *opacity*, *disable_type_checks: bool = False*)

Apply lighten only blending mode of a layer on an image.

### Example



```
import cv2, numpy
from blend_modes import lighten_only
img_in = cv2.imread('./orig.png', -1).astype(float)
img_layer = cv2.imread('./layer.png', -1).astype(float)
img_out = lighten_only(img_in,img_layer,0.5)
cv2.imshow('window', img_out.astype(numpy.uint8))
cv2.waitKey()
```

**See also:**

Find more information on Wikipedia.

**Parameters**

- **img_in** (*3-dimensional numpy array of floats (r/g/b/a) in range 0-255.0*) – Image to be blended upon
- **img_layer** (*3-dimensional numpy array of floats (r/g/b/a) in range 0.0-255.0*) – Layer to be blended with image
- **opacity** (*float*) – Desired opacity of layer for blending
- **disable_type_checks** (*bool*) – Whether type checks within the function should be disabled. Disabling the checks may yield a slight performance improvement, but comes at the cost of user experience. If you are certain that you are passing in the right arguments, you may set this argument to 'True'. Defaults to 'False'.

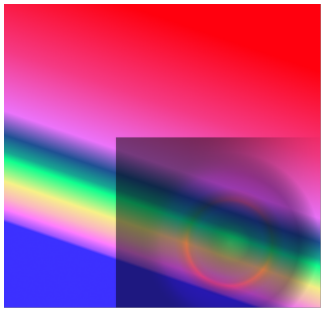
**Returns** Blended image

**Return type** 3-dimensional numpy array of floats (r/g/b/a) in range 0.0-255.0

**multiply**(*img_in*, *img_layer*, *opacity*, *disable_type_checks: bool = False*)

Apply multiply blending mode of a layer on an image.

### Example



```
import cv2, numpy
from blend_modes import multiply
img_in = cv2.imread('./orig.png', -1).astype(float)
img_layer = cv2.imread('./layer.png', -1).astype(float)
img_out = multiply(img_in,img_layer,0.5)
cv2.imshow('window', img_out.astype(numpy.uint8))
cv2.waitKey()
```

**See also:**

Find more information on Wikipedia.

**Parameters**

- **img_in** (*3-dimensional numpy array of floats (r/g/b/a) in range 0-255.0*) – Image to be blended upon
- **img_layer** (*3-dimensional numpy array of floats (r/g/b/a) in range 0.0-255.0*) – Layer to be blended with image
- **opacity** (*float*) – Desired opacity of layer for blending
- **disable_type_checks** (*bool*) – Whether type checks within the function should be disabled. Disabling the checks may yield a slight performance improvement, but comes at the cost of user experience. If you are certain that you are passing in the right arguments, you may set this argument to 'True'. Defaults to 'False'.

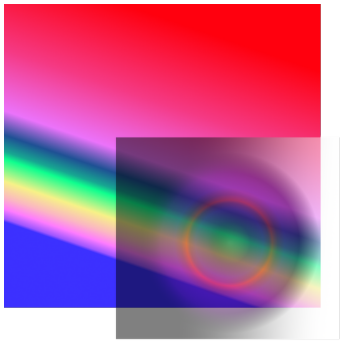
**Returns** Blended image

**Return type** 3-dimensional numpy array of floats (r/g/b/a) in range 0.0-255.0

**normal**(*img_in*, *img_layer*, *opacity*, *disable_type_checks: bool = False*)

Apply "normal" blending mode of a layer on an image.

### Example



```
import cv2, numpy
from blend_modes import normal
img_in = cv2.imread('./orig.png', -1).astype(float)
img_layer = cv2.imread('./layer.png', -1).astype(float)
img_out = normal(img_in,img_layer,0.5)
cv2.imshow('window', img_out.astype(numpy.uint8))
cv2.waitKey()
```

**See also:**

Find more information on Wikipedia.

**Parameters**

- **img_in** (*3-dimensional numpy array of floats (r/g/b/a) in range 0.0-255.0*) – Layer to be blended with image
- **img_layer** (*3-dimensional numpy array of floats (r/g/b/a) in range 0-255.0*) – Image to be blended upon
- **opacity** (*float*) – Desired opacity of layer for blending
- **disable_type_checks** (*bool*) – Whether type checks within the function should be disabled. Disabling the checks may yield a slight performance improvement, but comes at the cost of user experience. If you are certain that you are passing in the right arguments, you may set this argument to 'True'. Defaults to 'False'.

**Returns** Blended image

**Return type** 3-dimensional numpy array of floats (r/g/b/a) in range 0.0-255.0

**overlay**(*img_in*, *img_layer*, *opacity*, *disable_type_checks: bool = False*)

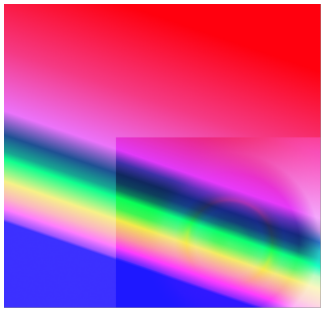
Apply overlay blending mode of a layer on an image.

---

**Note:** The implementation of this method was changed in version 2.0.0. Previously, it would be identical to the soft light blending mode. Now, it resembles the implementation on Wikipedia. You can still use the soft light blending mode if you are looking for backwards compatibility.

---

### Example



```
import cv2, numpy
from blend_modes import overlay
img_in = cv2.imread('./orig.png', -1).astype(float)
img_layer = cv2.imread('./layer.png', -1).astype(float)
img_out = overlay(img_in,img_layer,0.5)
cv2.imshow('window', img_out.astype(numpy.uint8))
cv2.waitKey()
```

**See also:**

Find more information on Wikipedia.

**Parameters**

- **img_in** (*3-dimensional numpy array of floats (r/g/b/a) in range 0-255.0*) – Image to be blended upon
- **img_layer** (*3-dimensional numpy array of floats (r/g/b/a) in range 0.0-255.0*) – Layer to be blended with image
- **opacity** (*float*) – Desired opacity of layer for blending
- **disable_type_checks** (*bool*) – Whether type checks within the function should be disabled. Disabling the checks may yield a slight performance improvement, but comes at the cost of user experience. If you are certain that you are passing in the right arguments, you may set this argument to 'True'. Defaults to 'False'.

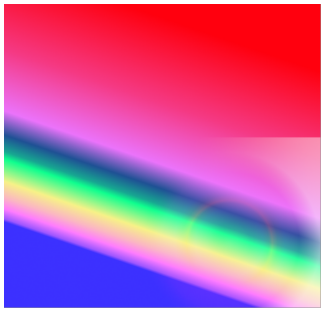
**Returns** Blended image

**Return type** 3-dimensional numpy array of floats (r/g/b/a) in range 0.0-255.0

**screen** (*img_in*, *img_layer*, *opacity*, *disable_type_checks: bool = False*)

Apply screen blending mode of a layer on an image.

### Example



```
import cv2, numpy
from blend_modes import screen
img_in = cv2.imread('./orig.png', -1).astype(float)
img_layer = cv2.imread('./layer.png', -1).astype(float)
img_out = screen(img_in,img_layer,0.5)
cv2.imshow('window', img_out.astype(numpy.uint8))
cv2.waitKey()
```

**See also:**

Find more information on Wikipedia.

**Parameters**

- **img_in** (*3-dimensional numpy array of floats (r/g/b/a) in range 0-255.0*) – Image to be blended upon
- **img_layer** (*3-dimensional numpy array of floats (r/g/b/a) in range 0.0-255.0*) – Layer to be blended with image
- **opacity** (*float*) – Desired opacity of layer for blending
- **disable_type_checks** (*bool*) – Whether type checks within the function should be disabled. Disabling the checks may yield a slight performance improvement, but comes at the cost of user experience. If you are certain that you are passing in the right arguments, you may set this argument to 'True'. Defaults to 'False'.

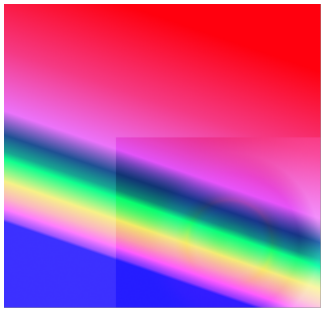
**Returns** Blended image

**Return type** 3-dimensional numpy array of floats (r/g/b/a) in range 0.0-255.0

**soft_light**(*img_in*, *img_layer*, *opacity*, *disable_type_checks: bool = False*)

Apply soft light blending mode of a layer on an image.

### Example



```
import cv2, numpy
from blend_modes import soft_light
img_in = cv2.imread('./orig.png', -1).astype(float)
img_layer = cv2.imread('./layer.png', -1).astype(float)
img_out = soft_light(img_in,img_layer,0.5)
cv2.imshow('window', img_out.astype(numpy.uint8))
cv2.waitKey()
```

**See also:**

Find more information on Wikipedia.

**Parameters**

- **img_in** (*3-dimensional numpy array of floats (r/g/b/a) in range 0-255.0*) – Image to be blended upon
- **img_layer** (*3-dimensional numpy array of floats (r/g/b/a) in range 0.0-255.0*) – Layer to be blended with image
- **opacity** (*float*) – Desired opacity of layer for blending
- **disable_type_checks** (*bool*) – Whether type checks within the function should be disabled. Disabling the checks may yield a slight performance improvement, but comes at the cost of user experience. If you are certain that you are passing in the right arguments, you may set this argument to 'True'. Defaults to 'False'.

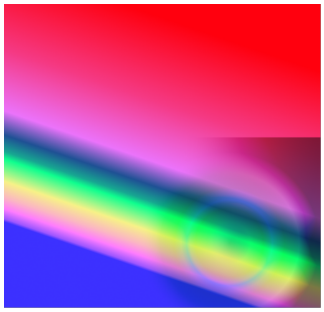
**Returns** Blended image

**Return type** 3-dimensional numpy array of floats (r/g/b/a) in range 0.0-255.0

**subtract**(*img_in*, *img_layer*, *opacity*, *disable_type_checks: bool = False*)

Apply subtract blending mode of a layer on an image.

### Example



```
import cv2, numpy
from blend_modes import subtract
img_in = cv2.imread('./orig.png', -1).astype(float)
img_layer = cv2.imread('./layer.png', -1).astype(float)
img_out = subtract(img_in,img_layer,0.5)
cv2.imshow('window', img_out.astype(numpy.uint8))
cv2.waitKey()
```

**See also:**

Find more information on Wikipedia.

**Parameters**

- **img_in** (*3-dimensional numpy array of floats (r/g/b/a) in range 0-255.0*) – Image to be blended upon
- **img_layer** (*3-dimensional numpy array of floats (r/g/b/a) in range 0.0-255.0*) – Layer to be blended with image
- **opacity** (*float*) – Desired opacity of layer for blending
- **disable_type_checks** (*bool*) – Whether type checks within the function should be disabled. Disabling the checks may yield a slight performance improvement, but comes at the cost of user experience. If you are certain that you are passing in the right arguments, you may set this argument to 'True'. Defaults to 'False'.

**Returns** Blended image

**Return type** 3-dimensional numpy array of floats (r/g/b/a) in range 0.0-255.0

CHAPTER 9

# Indices and Tables

- genindex
- modindex
- search

# Python Module Index

## b

# Index