Thumbor Documentation

Release 6.3.2

Bernardo Heynemann

May 22, 2017

Contents

thumbor.

CHAPTER 1

Whats Thumbor?

Thumbor is a smart imaging service. It enables on-demand crop, resizing and flipping of images.

It features a VERY smart detection of important points in the image for better cropping and resizing, using state-of-the-art face and feature detection algorithms (more on that in *Detection Algorithms*).

Using thumbor is very easy (after it is running). All you have to do is access it using an URL for an image, like this:

```
http://thumbor-server/unsafe/300x200/smart/s.glbimg.com/et/bb/f/original/2011/03/24/
↪VN0JiwzmOw0b0lg.jpg
```

That URL would show an image of the big brother brasil participants in 300x200 using smart crop. There are several other options to the image URL configuration. You can check them in the *Usage* page. For more details on the /unsafe part of the URL, check the *Security* page.

The safe url for the above URL would look like (check *Security* for more details):

```
http://thumbor-server/K97LekICOXT9MbO3X1u8BBkrjbu5/300x200/smart/s.glbimg.com/et/bb/f/
↪original/2011/03/24/VN0JiwzmOw0b0lg.jpg
```

CHAPTER 2

WARNING

Release 4.0.0 introduces a breaking change if you are using either the GraphicsMagick or the OpenCV Imaging engines. Please read the *Relase Notes* for more information.

CHAPTER 3

Contents

Installing

Installing thumbor is really easy because it supports the distutils form of packaging (http://docs.python.org/distutils/setupscript.html).

Stable

The latest stable version of thumbor is always published in the Python Package Index (http://pypi.python.org/pypi), so it can be easily installed using `pip install thumbor` or `easy_install thumbor`.

Ubuntu/Debian using aptitude (apt-get)

There's now an officially supported ppa for thumbor if you are using aptitude.

To install using aptitude, add the following lines to your sources list:

```
deb http://ppa.launchpad.net/thumbor/ppa/ubuntu <your release> main
deb-src http://ppa.launchpad.net/thumbor/ppa/ubuntu <your release> main
```

If you are using ubuntu 12.10 (quantal), it would be:

```
deb http://ppa.launchpad.net/thumbor/ppa/ubuntu quantal main
deb-src http://ppa.launchpad.net/thumbor/ppa/ubuntu quantal main
```

Or you can add the repository to you sources list via the command line:

```
sudo add-apt-repository ppa:thumbor/ppa
```

After that just update your sources:

```
sudo aptitude update
```

And install using plain old aptitude install:

```
sudo aptitude install thumbor
```

A service will be created for you that gets started when the machine starts up (using upstart).

By default thumbor will be disabled. Open `/etc/default/thumbor` and change (or remove) the flag `enabled` to `1` or use the command `sudo service thumbor start force=1` (force_start=1 for thumbor<3.7.0) to temporarily start thumbor. You can also override other defaults like the location of the configuration file by editing `/etc/default/thumbor`.

The configuration for thumbor will be at `/etc/thumbor.conf` and the security key at `/etc/thumbor.key`. There will be one instance running at `http://localhost:8888`.

If you want to run many instances of thumbor you'll need to run it in many ports. That means you'll need to use some form of load balancing (NGINX, Apache, Varnish, Haproxy, etc).

Running many instances of thumbor is as simple as editing `/etc/default/thumbor` and changing the `port` key to as many ports as you want, comma-separated: `port=8888,8889,8890` (for thumbor>3.7.0).

If you need more detail head to https://launchpad.net/~thumbor/+archive/ppa.

From the source of a stable release

Download the latest stable source-code version here on github or Pypi and decompress it.

In the path you decompressed, execute `pip install .` or `python setup.py install`.

From the latest version of the source

Clone thumbor's repository and install it using one of the following:

`pip install git+git://github.com/thumbor/thumbor.git`

or

```
git clone git://github.com/thumbor/thumbor.git

cd thumbor

python setup.py install
```

Alternative installation

Works on Ubuntu 12.04 x32, except webp detection with opencv doesn't seem to work :(

```
apt-get install ffmpeg libjpeg-dev libpng-dev libtiff-dev libjasper-dev libgtk2.0-dev␣
↪python-numpy python-pycurl webp python-opencv python-dev python-pip
pip install thumbor
```

Getting Started

If you just want to give thumbor a try, it is pretty easy to get started. **It won't take more than a minute.**

Just install it with `pip install thumbor` and start the process with `thumbor` in a console.

That's it! You are ready to start messing with images. Choose some image online and try the following.

If you can't install thumbor locally, but can use virtualbox, take a look at this repository: https://github.com/torchbox/vagrant-thumbor-base.

This tutorial assumes the image you picked is `http://www.waterfalls.hamilton.ca/images/Waterfall_Collage_home_sm1.jpg`. Replace it with whatever image you want accordingly.

Changing its size

Go to your browser and enter in the url: `http://localhost:8888/unsafe/300x200/http://www.waterfalls.hamilton.ca/images/Waterfall_Collage_home_sm1.jpg`.

You should see the waterfall image with 300px of width and 200px of height. Just play with it in the url to see the image change.

If you just want it to be proportional to the width, enter a height of 0, like:

`http://localhost:8888/unsafe/300x0/http://www.waterfalls.hamilton.ca/images/Waterfall_Collage_home_sm1.jpg`.

Flipping the image

How about seeing it backwards? Or upside down?

Go to your browser and enter in the url: `http://localhost:8888/unsafe/-0x-0/http://www.waterfalls.hamilton.ca/images/Waterfall_Collage_home_sm1.jpg`.

You should see the waterfall backwards and upside down.

Filters

What if I want to change contrast or brightness?

Go to your browser and enter in the url: `http://localhost:8888/unsafe/filters:brightness(10):contrast(30)/http://www.waterfalls.hamilton.ca/images/Waterfall_Collage_home_sm1.jpg`.

There are many more filters to explore. Check the *Filters* page for more details.

What now?

Ok, now that you know how amazing thumbor is, there's actually A LOT more to it. Go check the rest of the docs to learn how to get even more from your new imaging server.

Usage

Using thumbor is really straightforward. thumbor offers one endpoint for retrieving the image and a very similar endpoint to retrieve metadata.

Image Endpoint

**http://*thumbor-server*/hmac/trim/AxB:CxD/fit-in/-Ex-F/HALIGN/VALIGN/smart/filters:FILTERNAME(ARGUMENT
uri***

- *thumbor-server* is the address of the service currently running;
- hmac is the signature that ensures *Security* ;
- trim removes surrounding space in images using top-left pixel color unless specified otherwise;
- AxB:CxD means manually crop the image at left-top point AxB and right-bottom point CxD;
- fit-in means that the generated image should not be auto-cropped and otherwise just fit in an imaginary box specified by ExF;
- -Ex-F means resize the image to be ExF of width per height size. The minus signs mean flip horizontally and vertically;
- HALIGN is horizontal alignment of crop;
- VALIGN is vertical alignment of crop;
- smart means using smart detection of focal points;
- filters can be applied sequentially to the image before returning;
- *image-uri* is the public URI for the image you want resized.

Trim

Removing surrounding space in images can be done using the trim option.

Unless specified trim assumes the top-left pixel color and no tolerance (more on tolerance below).

To use it, just add a `/trim` part to your URL.

If you need to specify the orientation from where to get the pixel color, just use `/trim:top-left` for the top-left pixel color or `/trim:bottom-right` for the bottom-right pixel color.

Trim also supports color tolerance. The euclidian distance between the colors of the reference pixel and the surrounding pixels is used. If the distance is within the tolerance they'll get trimmed. For a RGB image the tolerance would be within the range 0-442.

Manual Crop

The manual crop is entirely optional. This is very useful for applications that provide custom real-time cropping capabilities to their users.

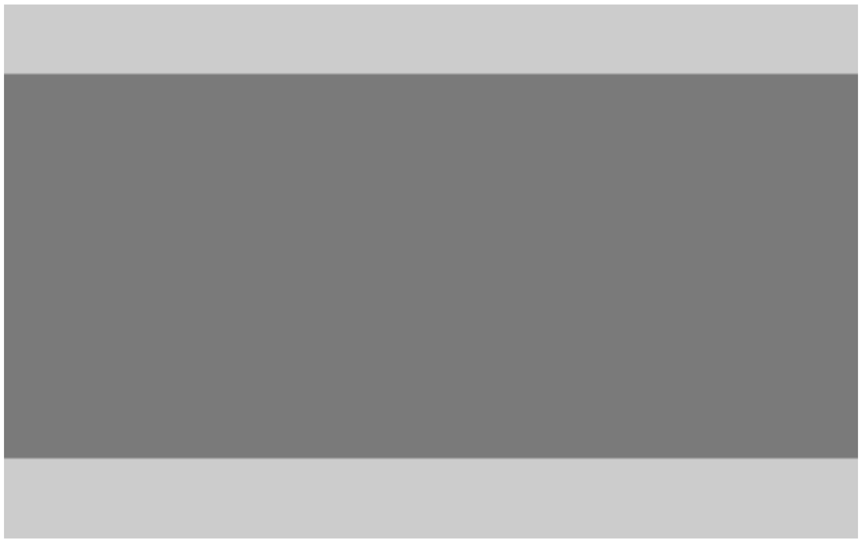
The manual crop part of the url takes two points as arguments, separated by a colon. The first point is the left-top point of the cropping rectangle. The second point is the right-bottom point.

This crop is performed before the rest of the operations, so it can be used as a *prepare* step before resizing and smart-cropping. It is **very** useful when you just need to get that celebrity face on a big picture full of people, as an example.

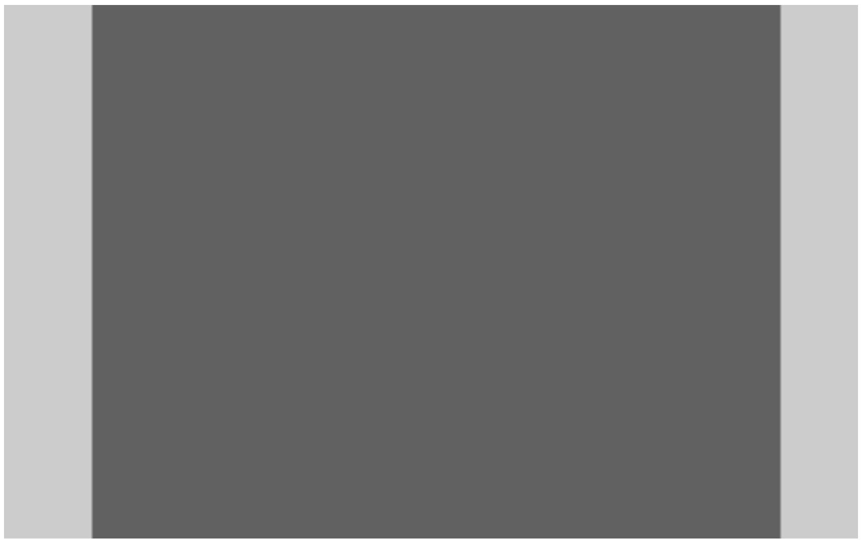
Fit in

The fit-in argument specifies that the image should not be auto-cropped but auto-resized (shrinked) to fit in an imaginary box of "E" width and "F" height, instead.

Consider an image of 800px x 600px, and a fit of 300px x 200px. This is how thumbor would resize it:



Consider an image of 400px x 600px, and a fit of 300px x 200px. This is how thumbor would resize it:



This is very useful when you need to fit an image somewhere, but you have no idea about the original image dimensions.

Keep in mind that it won't enlarge your image in case it is smaller than the specified "E" width and "F" height! One way of getting the image in the size requested is to use the *Filling* filter.

Image Size

The image size argument specifies the size of the image that will be returned by the service. Thumbor uses smart *Crop and Resize Algorithms*

If you omit one of the dimensions or use zero as a value (as in 300x, 300x0, x200, 0x200, and so on), Thumbor will determine that dimension as to be proportional to the original image. Say you have an 800x600 image and ask for a 400x0 image. Thumbor will infer that since 400 is half of 800, then the height you are looking for is half of 600, which is 300px.

If you use 0x0, Thumbor will use the original size of the image and thus won't do any cropping or resizing.

If you specify one of the dimensions as the string "orig" (as in origx100, 100xorig, origxorig), thumbor will interpret that you want that dimension to remain the same as in the original image. Consider an image of 800x600. If you ask for a 300xorig version of it, thumbor will interpret that you want a 300x600 image. If you instead ask for a origx300 version, thumbor will serve you an 800x300 image.

If you use origxorig, Thumbor will use the original size of the image and thus won't do any cropping or resizing.

The default value (in case it is omitted) for this option is to use proportional size (0) to the original image.

Horizontal Align

As was explained above, unless the image is of the same proportion as the desired size, some cropping will need to occur.

The horizontal align option controls where the cropping will occur if some width needs to be trimmed (unless some feature detection occurs - more on that later).

So, if we need to trim 300px of the width and the current horizontal align is "left", then we'll trim 0px of the left of the image and 300px of the right side of the image.

The possible values for this option are:

- left - only trims the right side;
- center - trims half of the width from the left side and half from the right side;
- right - only trims the left side.

It is important to notice that this option is useless in case of the image being vertically trimmed, since Thumbor's cropping algorithm only crops in one direction.

The default value (in case it is omitted) for this option is "center".

Vertical Align

The vertical align option is analogous to the horizontal one, except that it controls height trimming.

So, if we need to trim 300px of the height and the current vertical align is "top", then we'll trim 0px of the top of the image and 300px of the bottom side of the image.

The possible values for this option are:

- top - only trims the bottom;
- middle - trims half of the height from the top and half from the bottom;
- bottom - only trims the top.

It is important to notice that this option is useless in case of the image being horizontally trimmed, since Thumbor's cropping algorithm only crops in one direction.

The default value (in case it is omitted) for this option is "middle".

Smart Cropping

Thumbor uses some very advanced techniques for obtaining important points of the image (referred to as Focal Points in the rest of this documentation).

Even though Thumbor comes with facial recognition of Focal Points as well as feature recognition, you can easily implement your own detectors as you'll see further in the docs.

There's not much to this option, since we'll cover it in the *Detection Algorithms* page. If you use it in the url, smart cropping will be performed and will override both horizontal and vertical alignments if it finds any Focal Points.

The default value (in case it is omitted) for this option is not to use smart cropping.

Filters

Thumbor allows for usage of a filter pipeline that will be applied sequentially to the image. Filters are covered in the *Filters* page if you want to know more.

To use filters add a "filters:" part in your URL. Filters are like function calls "filter_name(argument, argument2, etc)" and are separated using the ':' character.

Image URI

This is the image URI. The format of this option depends heavily on the image loader you are using. Thumbor comes pre-packaged with an HTTP loader and a Filesystem loader.

If you use the HTTP loader, this option corresponds to the image complete URI.

If you use the Filesystem loader, this option corresponds to the path of the image from the images root.

You can learn more about the loaders in the *Image loader* page.

Metadata Endpoint

The metadata endpoint has **ALL** the options that the image one has, but instead of actually performing the operations in the image, it just simulates the operations.

Since it has the same options as the other endpoint, we won't repeat all of them. To use the metadata endpoint, just add a */meta* in the beggining of the url.

Say we have the following crop URL:

http://my-server.thumbor.org/unsafe/-300x-200/left/top/smart/path/to/my/nice/image.jpg

If we want the metadata on what thumbor would do, just change the url to be

http://my-server.thumbor.org/unsafe/meta/-300x-200/left/top/smart/path/to/my/nice/image.jpg

After the processing is finished, thumbor will return a json object containing metadata on the image and the operations that would have been performed.

The json looks like this:

```
{
    thumbor: {
        source: {
            url: "path/to/my/nice/image.jpg",
            width: 800,
            height: 600
        },
        operations: [
            {
                type: "crop",
                left: 10,
                top: 10,
                right: 300,
                bottom: 200
            },
            {
                type: "resize",
                width: 300,
                height: 200
            },
            { type: "flip_horizontally" },
            { type: "flip_vertically" }
        ]
    }
}
```

Imaging

Detectors

Enabling detectors

Out of the box, thumbor does not enable any feature or facial detection. Enabling it is pretty easy, though.

This documentation assumes you have OpenCV installed. It is a requirement if you want to use thumbor's *Available detectors*.

Configuration

In order to tell thumbor what detectors it should run in the original image, you must add them to your `thumbor.conf` file in the following key:

```
DETECTORS = [
    'thumbor.detectors.face_detector',
    'thumbor.detectors.feature_detector',
]
```

The above configuration tells thumbor that it should run both the facial detection and the feature detection. These are mutually exclusive, meaning that if a face is detected, the feature detector won't be run.

Using it

After restarting thumbor, it should be as easy as adding a `/smart` option to your URLs, like:

```
http://<thumbor>:<port>/unsafe/300x200/smart/my.domain.com/picture.png
```

Lazy Detection

Facial detection can be pretty expensive for thumbor, so it is not advisable to do it synchronously. Please refer to the *Lazy Detection* page for instructions on using it.

Available Detectors

A list of available detectors can be found at *Available detectors*.

Available detectors

Face Detector

`thumbor.detectors.face_detector`

TBW.

Feature Detector

`thumbor.detectors.feature_detector`

TBW.

Glasses Detector

`thumbor.detectors.glasses_detector`

TBW.

Profile Detector

`thumbor.detectors.profile_detector`

TBW.

Queued Detector

`thumbor.detectors.queued_detector`

TBW.

Custom detection

If you need more detection than the pre-packaged detectors are able to give you (i.e.: you need to detect glasses), you can always implement your own detectors.

If your detector can be found using python's import mechanism, thumbor will be able to use it. Just add its full name to the detectors *Configuration*.

Creating a Custom Detector

As you can see here https://github.com/thumbor/thumbor/blob/master/thumbor/detectors/face_detector/__init__.py it is pretty easy to implement your own custom detector.

All you have to do is create a class that inherits from BaseDetector and implement a detect method that receives a context dictionary.

In the context dictionary there's a key called "focal_points" to which you should append any focal points you found in the picture (using the FocalPoint class).

If your detector does not find any points, simple call the next() method passing in the context, so further detection can occur.

Filters

Attention: Filters are affecting each other in the order they are specified! If the original image has the size `60x40` and the thumbor url would look like:

```
fit-in/100x100/filters:watermark(..):blur(..):fill(red,
1):upscale()/http://some/image.jpeg
```

then the image will first checked if it fits into `100x100` (which it does), then gets the watermark, then it will be blurred (including the watermark), then filled with red so that it will use the `100x100` size and at the end upscaled will be applied without having any effect because fill was changing the size of the image already to the maximum available space.

Background Color

Usage: background_color(color)

Description

The background_color filter sets the background layer to the specified color. This is specifically useful when converting transparent images (PNG) to JPEG

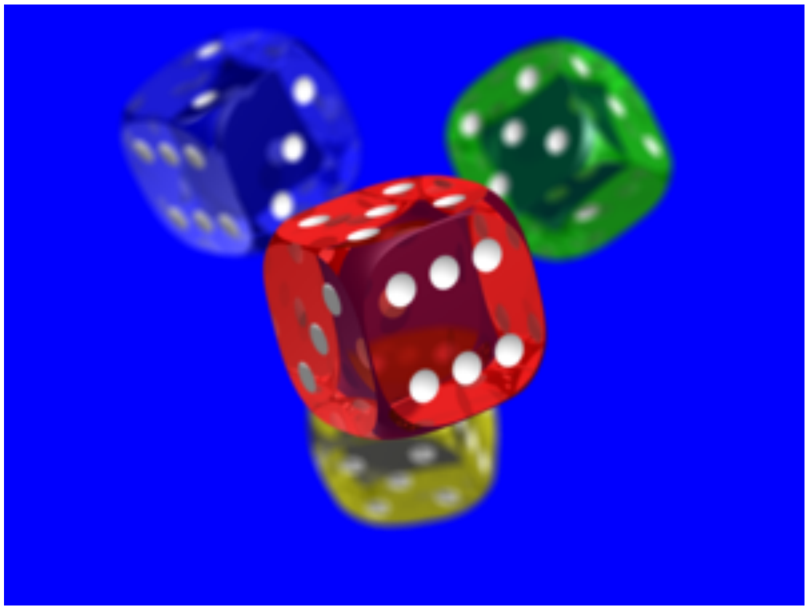
Arguments

- color - the color name (like in HTML) or hexadecimal rgb expression without the "#" character (see https://en.wikipedia.org/wiki/Web_colors for example). If color is "auto", a color will be smartly chosen (based on the image pixels) to be the filling color.

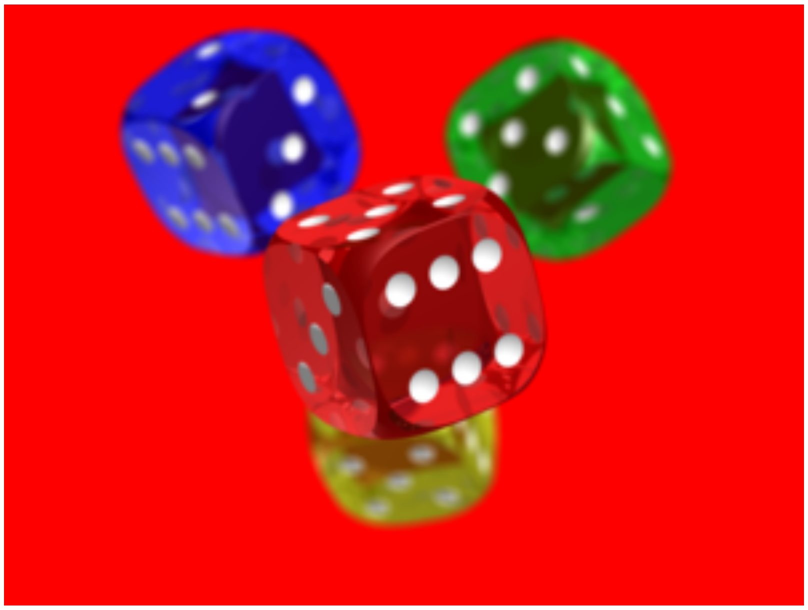
Example

The original image is:

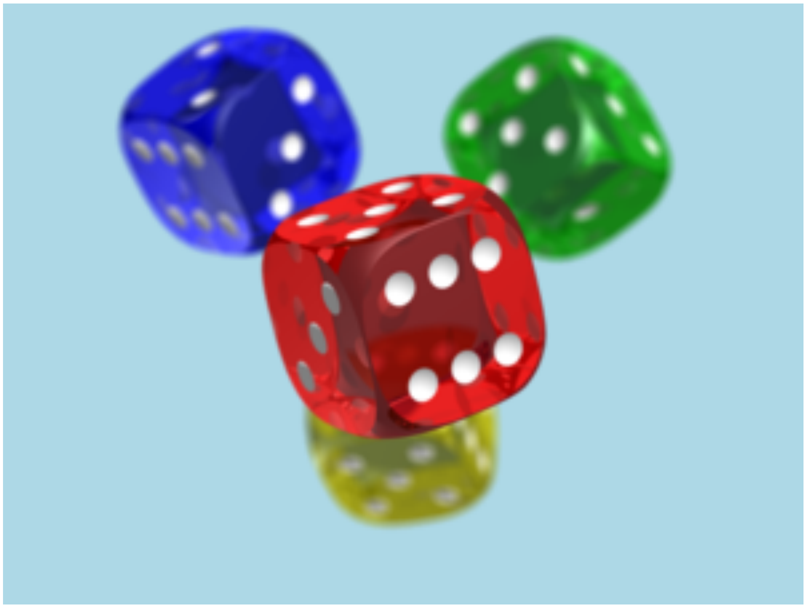
http://localhost:8888/unsafe/fit-in/300x300/filters:background_color(blue)/https://github.com/thumbor/thumbor/wiki/dice_transparent_background.png



http://localhost:8888/unsafe/fit-in/300x300/filters:background_color(f00)/https://github.com/thumbor/thumbor/wiki/dice_transparent_background.png



http://localhost:8888/unsafe/fit-in/300x300/filters:background_color(add8e6)/https://github.com/thumbor/thumbor/wiki/dice_transparent_background.png



Blur

Usage: blur(radius [, sigma])

Description

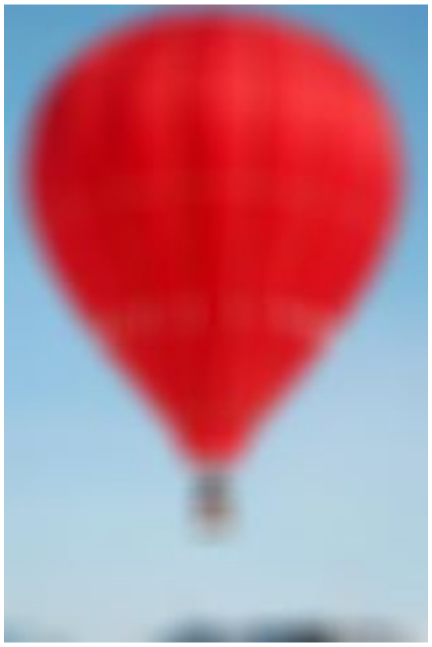
This filter applies a gaussian blur to the image.

Arguments

- radius - Radius used in the gaussian function to generate a matrix, maximum value is 150. The bigger the radius more blurred will be the image.
- sigma - Optional. Defaults to the same value as the radius. Sigma used in the gaussian function.

Example

http://localhost:8888/unsafe/filters:blur(7)/http://upload.wikimedia.org/wikipedia/commons/thumb/8/8a/2006_Ojiya_balloon_festival_011.jpg/159px-2006_Ojiya_balloon_festival_011.jpg



Brightness

Usage: brightness(amount)

Description

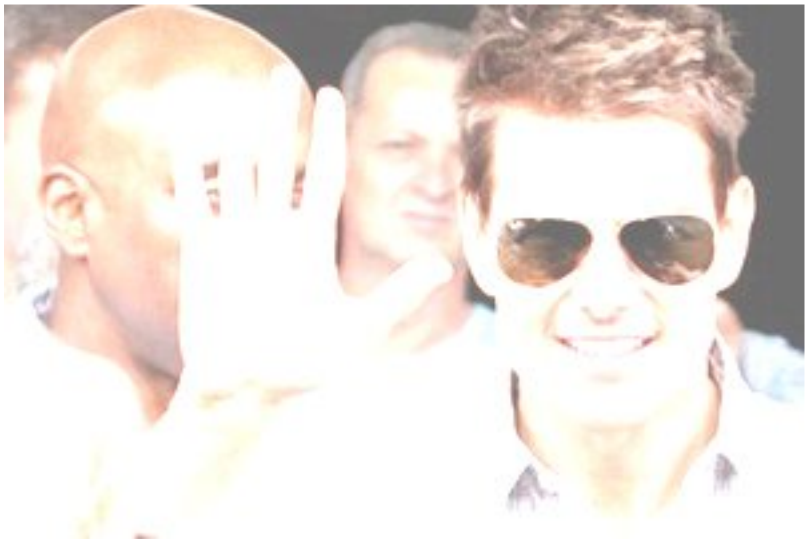
This filter increases or decreases the image brightness.

Arguments

amount - -100 to 100 - The amount (in %) to change the image brightness. Positive numbers make the image brighter and negative numbers make the image darker.

Example

http://thumbor-server/filters:brightness(40)/some/image.jpg



Contrast

Usage: contrast(amount)

Description

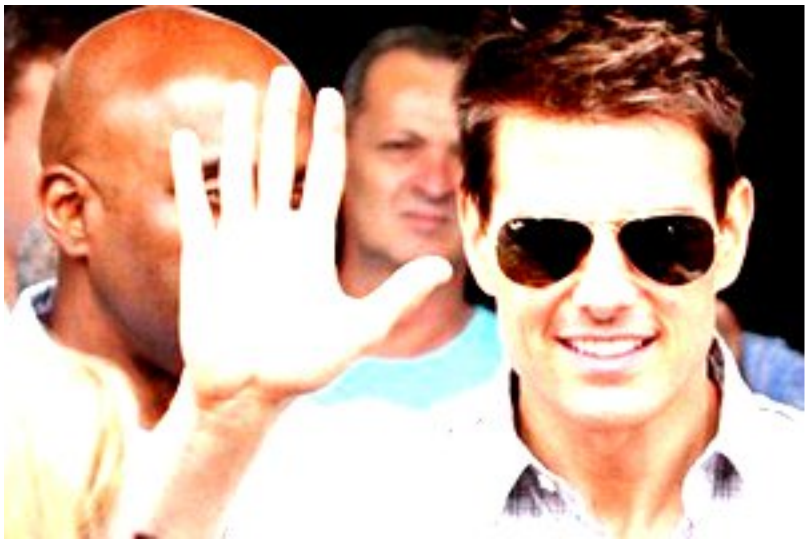
This filter increases or decreases the image contrast.

Arguments

amount - -100 to 100 - The amount (in %) to change the image contrast. Positive numbers increase contrast and negative numbers decrease contrast.

Example

http://thumbor-server/filters:contrast(40)/some/image.jpg



http://thumbor-server/filters:contrast(-40)/some/image.jpg

Convolution

Usage: convolution(matrix_items, number_of_columns, should_normalize)

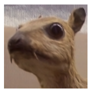
Description

This filter runs a convolution matrix (or kernel) on the image. See Kernel (image processing) for details on the process. Edge pixels are always extended outside the image area.

Arguments

- matrix_items - Semicolon separated matrix items.
- number_of_columns - Number of columns in the matrix.
- should_normalize - Whether or not we should divide each matrix item by the sum of all items.

Example



Normalized Matrix:

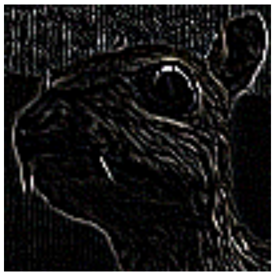
```
1 2 1
2 4 2
2 1 2
```

http://localhost:8888/unsafe/filters:convolution(1;2;1;2;4;2;1;2;1,3,true)/http://upload.wikimedia.org/wikipedia/commons/5/50/Vd-Orig.png

Matrix:

```
-1 -1 -1
-1  8 -1
-1 -1 -1
```

http://localhost:8888/unsafe/filters:convolution(-1;-1;-1;-1;8;-1;-1;-1;-1,3,false)/http://upload.wikimedia.org/wikipedia/commons/5/50/Vd-Orig.png



Equalize

Usage: equalize()

Description

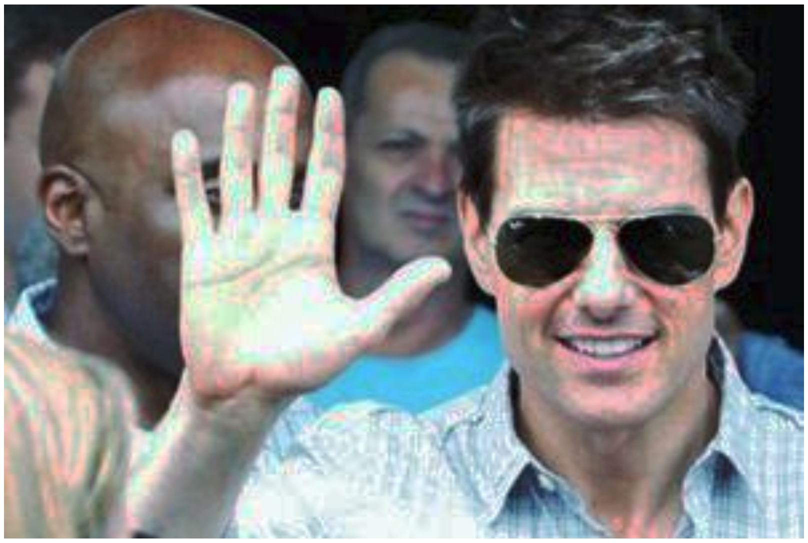
This filter equalizes the color distribution in the image.

Arguments

No arguments.

Example

http://thumbor-server/filters:equalize()/some/image.jpg



Extract focal points

Usage: extract_focal()

Description

When cropping, thumbor uses focal points in the image to direct the area of the image that matters most. There are several ways of finding focal points. To learn more about focal points, visit the *Detection Algorithms*.

In order to use the `extract_focal` filter, the original image must be a thumbor URL that features manual cropping. To learn more about manual cropping, visit the *Crop and Resize Algorithms*.

Using the original manual cropping points, this filter adds the cropped area (originally in the format */LEFTx-TOP:RIGHTxBOTTOM/*) as a focal point for the new image.

For the new image, thumbor will use as the original the image URL that was the original for the segment with the manual cropping.

This means that for an URL like:

http://localhost:8888/unsafe/300x100/filters:extract_focal()/localhost:8888/unsafe/100x150:300x200/https://upload.wikimedia.org/wikipedia/commons/thumb/2/22/Turkish_Van_Cat.jpg/546px-Turkish_Van_Cat.jpg

Thumbor will use as original the following image URL:

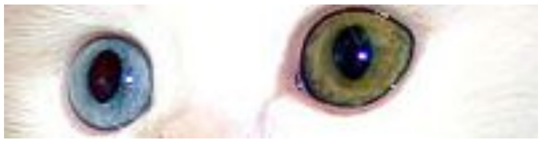
https://upload.wikimedia.org/wikipedia/commons/thumb/2/22/Turkish_Van_Cat.jpg/546px-Turkish_Van_Cat.jpg

Example

Original Image:

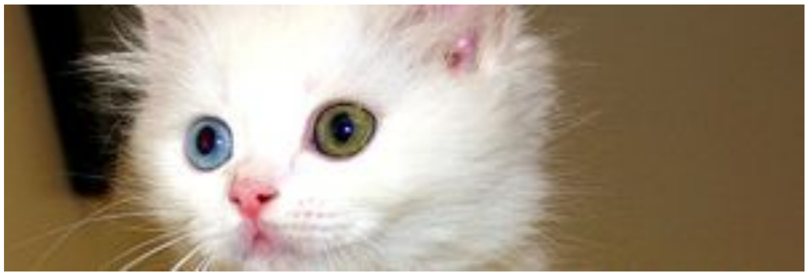
Cat's eye cropped:

http://localhost:8888/unsafe/100x150:300x200/https://upload.wikimedia.org/wikipedia/commons/thumb/2/22/Turkish_Van_Cat.jpg/546px-Turkish_Van_Cat.jpg



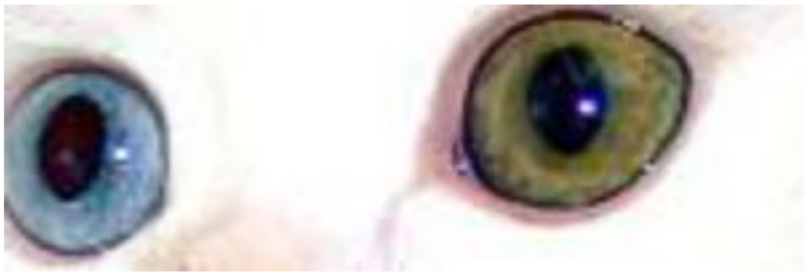
A bigger image based on above's crop with the extract_focal() filter:

http://localhost:8888/unsafe/300x100/filters:extract_focal()/localhost:8888/unsafe/100x150:300x200/https://upload.wikimedia.org/wikipedia/commons/thumb/2/22/Turkish_Van_Cat.jpg/546px-Turkish_Van_Cat.jpg



Without the filter that would be the result:

http://localhost:8888/unsafe/300x100/localhost:8888/unsafe/100x150:300x200/https://upload.wikimedia.org/wikipedia/commons/thumb/2/22/Turkish_Van_Cat.jpg/546px-Turkish_Van_Cat.jpg



Filling

Usage: *fill(color[,fill_transparent])*

Description

This filter permit to return an image sized exactly as requested wherever is its ratio by filling with chosen color the missing parts. Usually used with “fit-in” or “adaptive-fit-in”

Arguments

- color - the color name (like in HTML) or hexadecimal RGB expression without the “#” character (see https://en.wikipedia.org/wiki/Web_colors for example).

  If color is “transparent” and the image format, supports transparency the filling color is transparent[1].

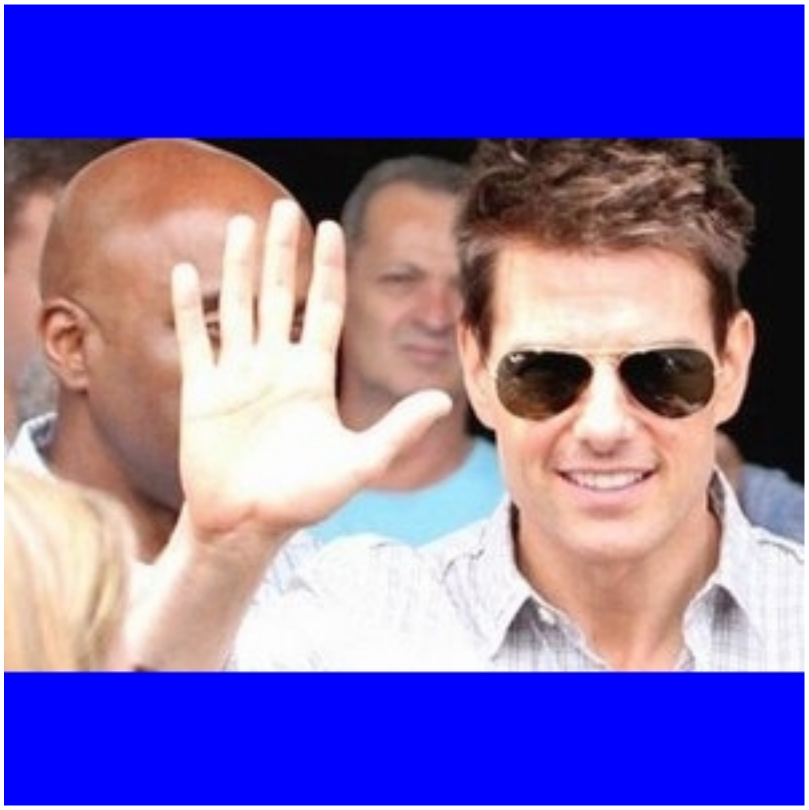
  If color is “auto”, a color is smartly chosen (based on the image pixels) as the filling color.
- fill_transparent - a boolean to specify whether transparent areas of the image should be filled or not. Accepted values are either *true*, *false*, *1* or *0*. This argument is optional and the default value is *false*.

Example #1

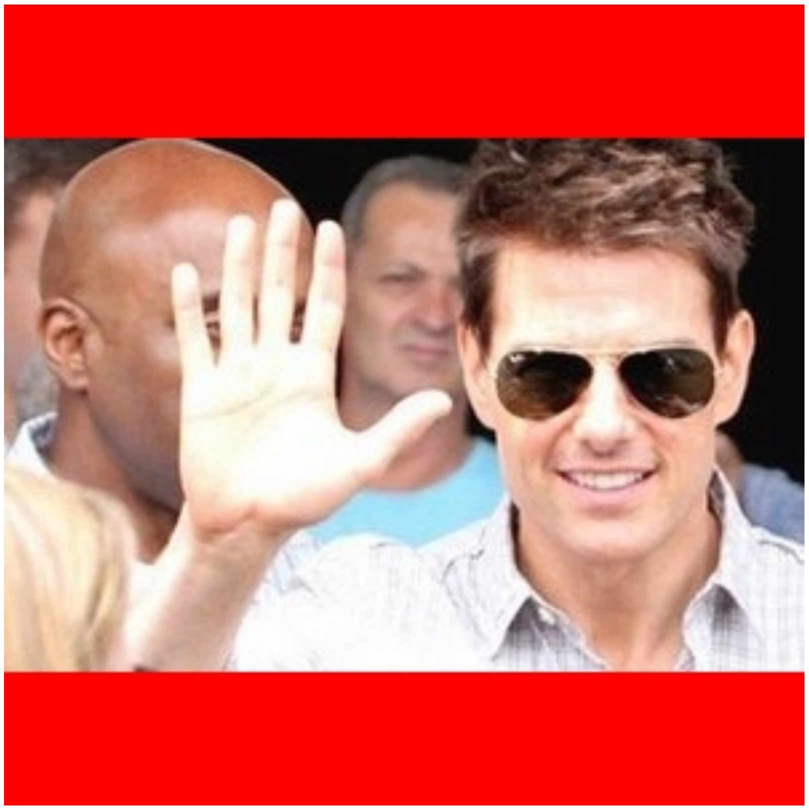
The original image is:

http://localhost:8888/unsafe/fit-in/300x300/filters:fill(blue)/https://github.com/thumbor/thumbor/wiki/tom_before_brightness.jpg

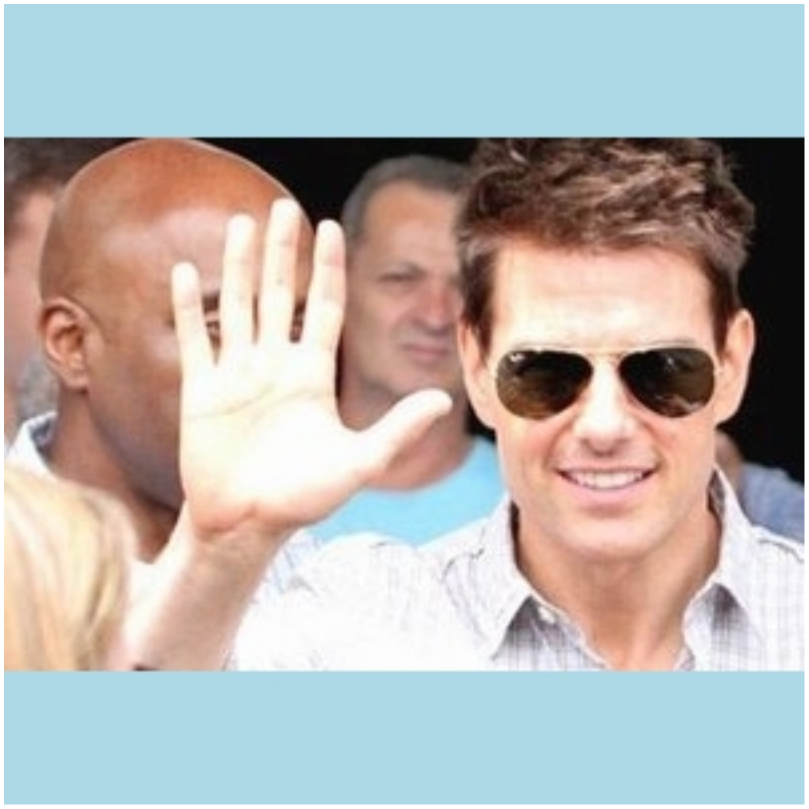
[1] OpenCV Engine does not support transparent filling



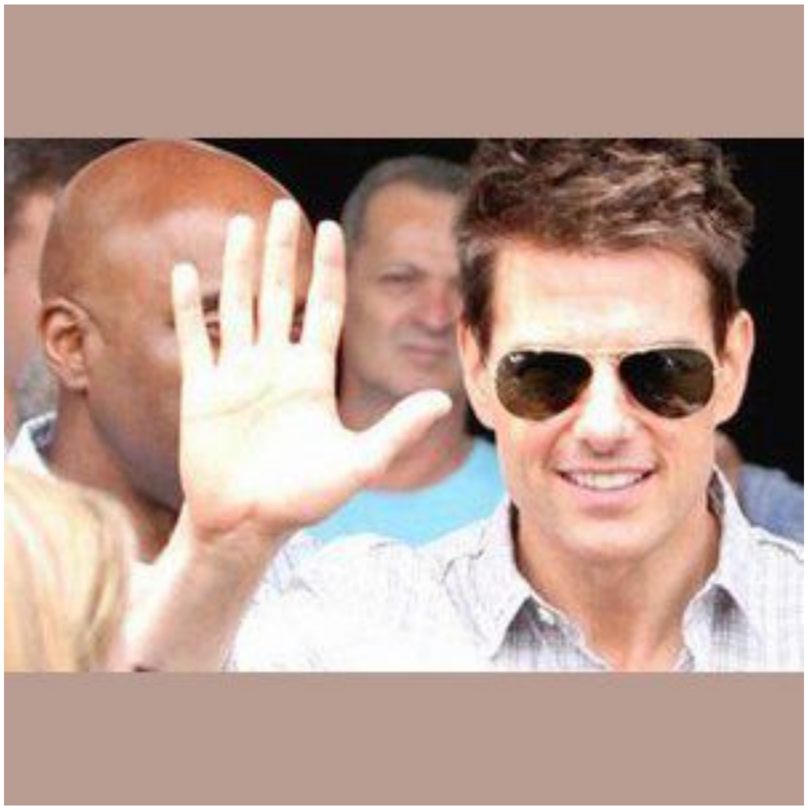
http://localhost:8888/unsafe/fit-in/300x300/filters:fill(f00)/https://github.com/thumbor/thumbor/wiki/tom_before_
brightness.jpg



http://localhost:8888/unsafe/fit-in/300x300/filters:fill(add8e6)/https://github.com/thumbor/thumbor/wiki/tom_before_brightness.jpg

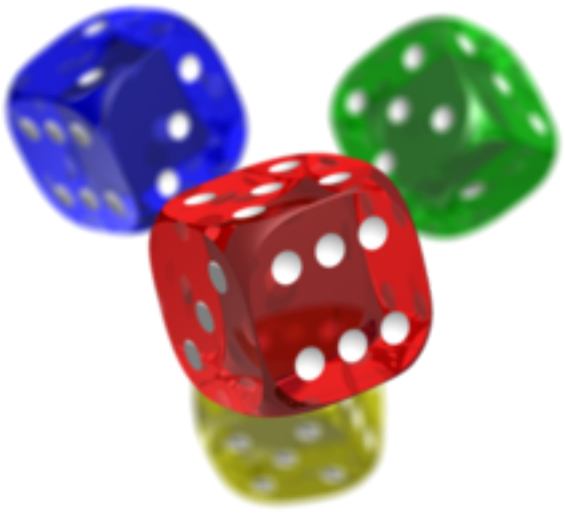


http://localhost:8888/unsafe/fit-in/300x300/filters:fill(auto)/https://github.com/thumbor/thumbor/wiki/tom_before_brightness.jpg



Example #2

The original image is:



http://localhost:8888/unsafe/fit-in/300x225/filters:fill(blue,1)/https://github.com/thumbor/thumbor/wiki/dice_transparent_background.png

http://localhost:8888/unsafe/fit-in/300x225/filters:fill(f00,true)/https://github.com/thumbor/thumbor/wiki/dice_transparent_background.png

http://localhost:8888/unsafe/fit-in/300x225/filters:fill(add8e6,1)/https://github.com/thumbor/thumbor/wiki/dice_transparent_background.png

http://localhost:8888/unsafe/fit-in/300x225/filters:fill(auto,true)/https://github.com/thumbor/thumbor/wiki/dice_transparent_background.png

Focal

Usage: focal(<left>x<top>:<right>x<bottom>)

Description

This filter adds a focal point, which is used in later transforms.

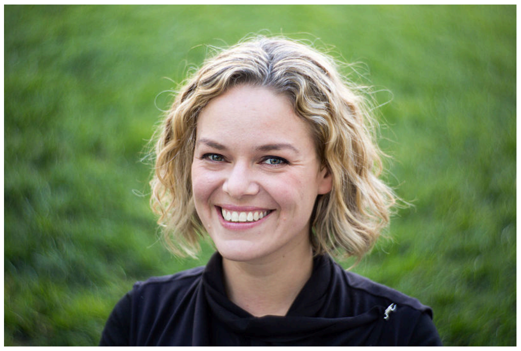
Arguments

- left, top, right, bottom: All mandatory arguments in the `<left>x<top>:<right>x<bottom>` format.

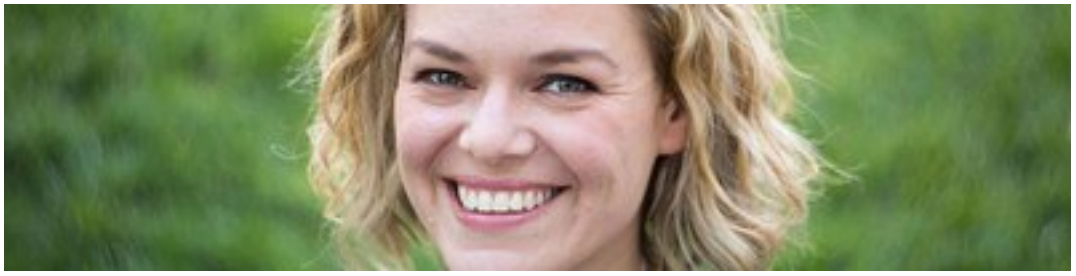
Example

```
http://localhost:8888/unsafe/400x100/filters:focal(146x206:279x360)/https:/
/upload.wikimedia.org/wikipedia/commons/thumb/2/25/Katherine_Maher.jpg/
800px-Katherine_Maher.jpg
```

Before



After



Warning

Unpredictable behavior may occur if using this filter with other functionality that set focal points, such as:

Image Metadata

Thumbor uses Pyexiv2 to read and write image metadata.

If the Pyexif2 or Py3exif2 Python library is available, the PIL engine also stores image metadata in `engine.metadata`.

Reading and writing Metadata

This part is copied from the Pyexiv2 Tutorial

Let's retrieve a list of all the available EXIF tags available in the image:

```
>>> engine.metadata.exif_keys
['Exif.Image.ImageDescription',
 'Exif.Image.XResolution',
 'Exif.Image.YResolution',
 'Exif.Image.ResolutionUnit',
 'Exif.Image.Software',
 'Exif.Image.DateTime',
 'Exif.Image.Artist',
 'Exif.Image.Copyright',
 'Exif.Image.ExifTag',
 'Exif.Photo.Flash',
 'Exif.Photo.PixelXDimension',
 'Exif.Photo.PixelYDimension']
```

Each of those tags can be accessed with the `[]` operator on the metadata, much like a python dictionary:

```
>>> tag = metadata[b'Exif.Image.DateTime']
```

The value of an `ExifTag` object can be accessed in two different ways: with the `raw_value` and with the `value` attributes:

```
>>> tag.raw_value
'2004-07-13T21:23:44Z'

>>> tag.value
datetime.datetime(2004, 7, 13, 21, 23, 44)
```

The raw value is always a byte string, this is how the value is stored in the file. The value is lazily computed from the raw value depending on the EXIF type of the tag, and is represented as a convenient python object to allow easy manipulation.

Note that querying the value of a tag may raise an `ExifValueError` if the format of the raw value is invalid according to the EXIF specification (may happen if it was written by other software that implements the specification in a broken manner), or if pyexiv2 doesn't know how to convert it to a convenient python object.

Accessing the value of a tag as a python object allows easy manipulation and formatting:

```
>>> tag.value.strftime('%A %d %B %Y, %H:%M:%S')
'Tuesday 13 July 2004, 21:23:44'
```

Now let's modify the value of the tag and write it back to the file:

```
>>> import datetime
>>> tag.value = datetime.datetime.today()

>>> engine.metadata.write()
```

Similarly to reading the value of a tag, one can set either the `raw_value` or the `value` (which will be automatically converted to a correctly formatted byte string by pyexiv2).

You can also add new tags to the metadata by providing a valid key and value pair (see exiv2's documentation for a list of valid EXIF tags):

```
>>> key = 'Exif.Photo.UserComment'
>>> value = 'This is a useful comment.'
>>> engine.metadata[key] = pyexiv2.ExifTag(key, value)
```

As a handy shortcut, you can always assign a value for a given key regardless of whether it's already present in the metadata. If a tag was present, its value is overwritten. If the tag was not present, one is created and its value is set:

```
>>> engine.metadata[key] = value
```

The EXIF data may optionally embed a thumbnail in the JPEG or TIFF format. The thumbnail can be accessed, set from a JPEG file or buffer, saved to disk and erased:

```
>>> thumb = engine.metadata.exif_thumbnail
>>> thumb.set_from_file('/tmp/thumbnail.jpg')
>>> thumb.write_to_file('/tmp/copy')
>>> thumb.erase()
>>> engine.metadata.write()
```

Installation

Pyexiv2 depends on the following libraries:

- boost.python (http://www.boost.org/libs/python/doc/index.html)
- exiv2 (http://www.exiv2.org/)

On OSX you can use homebrew to install the dependencies:

```
brew install boost --with-python
brew install boost-python
brew install exiv2

pip install git+https://github.com/escaped/pyexiv2.git
```

If you are updating thumbor and already have an existing virtualenv, then you have to recreate it. If you have both a System Python and a Homebrew Python with the same version, then make sure the Virtualenv uses the Homebrew Python binary.

On Linux Pyexiv2 can be installed with apt-get:

apt-get install python-pyexiv2

pyexiv2.metadata API reference

Currently PyExiv is deprecated in favor of GExiv. However, it is really difficult to install GExiv with Python on a non-Ubuntu system. Therefore Pyexiv2 is used.

Format

Usage: format(image-format)

Description

This filter specifies the output format of the image. The output must be one of: "webp", "jpeg", "gif" or "png".

Arguments

image-format - The output format of the resulting image.

Example

http://thumbor-server/filters:format(webp)/some/image.jpg

Grayscale

Usage: grayscale()

Description

This filter changes the image to grayscale.

Arguments

No arguments.

Example

http://thumbor-server/filters:grayscale()/some/image.jpg

Max bytes

Usage: max_bytes(number-of-bytes)

Description

This filter automatically degrades the quality of the image until the image is under the specified amount of bytes.

Arguments

number-of-bytes - The maximum number of bytes for the given image.

Example

http://thumbor-server/filters:max_bytes(40000)/some/image.jpg

No upscale

Usage: no_upscale()

Description

This filter tells thumbor not to upscale your images.

This means that if an original image is 300px width by 200px height and you ask for a 600x400 image, thumbor will still return a 300x200 image.

Arguments

No arguments allowed.

Example

http://thumbor-server/unsafe/300x200/filters:no_upscale()/some/image.jpg

Noise

Usage: noise(amount)

Description

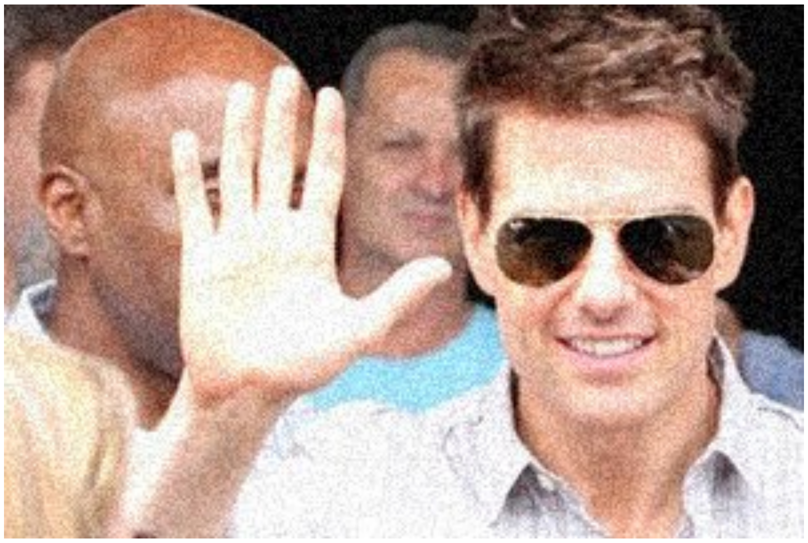
This filter adds noise to the image.

Arguments

amount - 0 to 100 - The amount (in %) of noise to add to the image.

Example

http://thumbor-server/filters:noise(40)/some/image.jpg



Quality

Usage: quality(amount)

Description

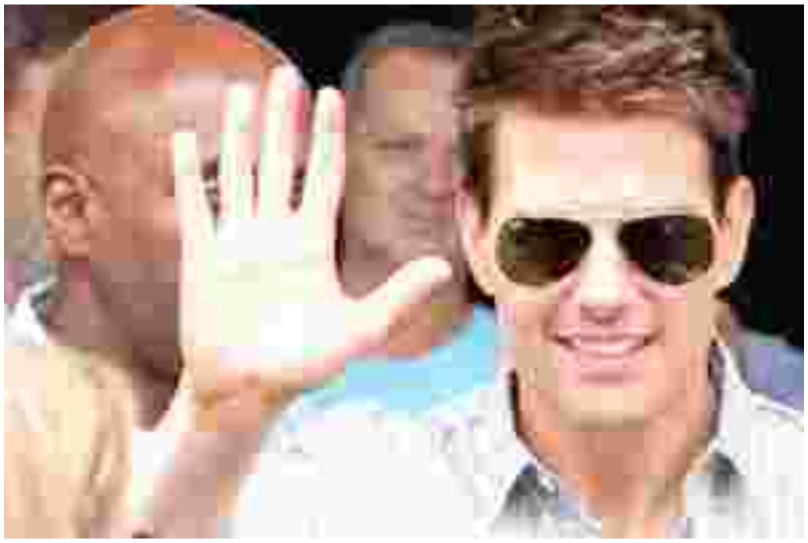
This filter changes the overall quality of the JPEG image (does nothing for PNGs or GIFs).

Arguments

amount - 0 to 100 - The quality level (in %) that the end image will feature.

Example

http://thumbor-server/filters:quality(40)/some/image.jpg



Red eye

Not documented yet

RGB

Usage: rgb(rAmount, gAmount, bAmount)

Description

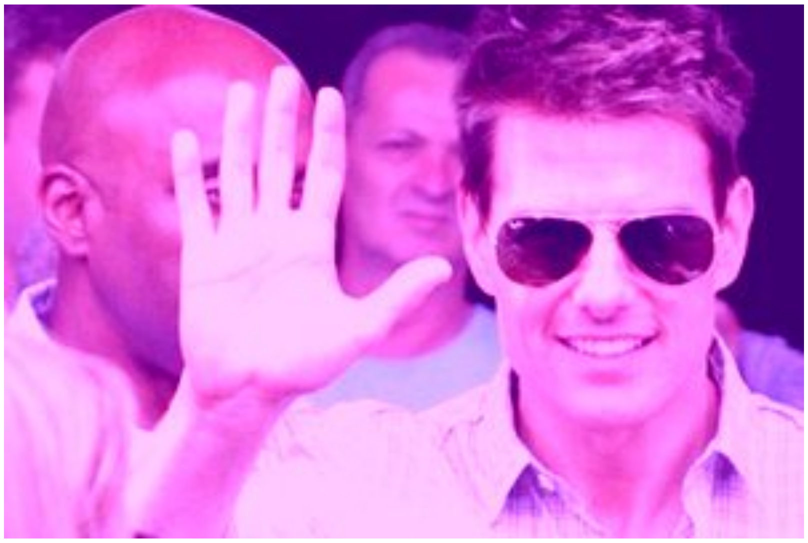
This filter changes the amount of color in each of the three channels.

Arguments

- rAmount - The amount of redness in the picture. Can range from -100 to 100 in percentage.
- gAmount - The amount of greenness in the picture. Can range from -100 to 100 in percentage.
- bAmount - The amount of blueness in the picture. Can range from -100 to 100 in percentage.

Example

http://thumbor-server/filters:rgb(20,-20,40)/some/image.jpg



Rotate

Usage: rotate(angle)

Description

This filter rotate the given image according to the angle value passed.

Arguments

angle - 0 to 359 - The angle to rotate the image. Numbers greater or equal than 360 will be tranformed to a equivalent angle between 0 and 359.

Example

http://thumbor-server/filters:rotate(90)/some/image.jpg

Round corners

Usage: round_corner(a|b,r,g,b)

Description

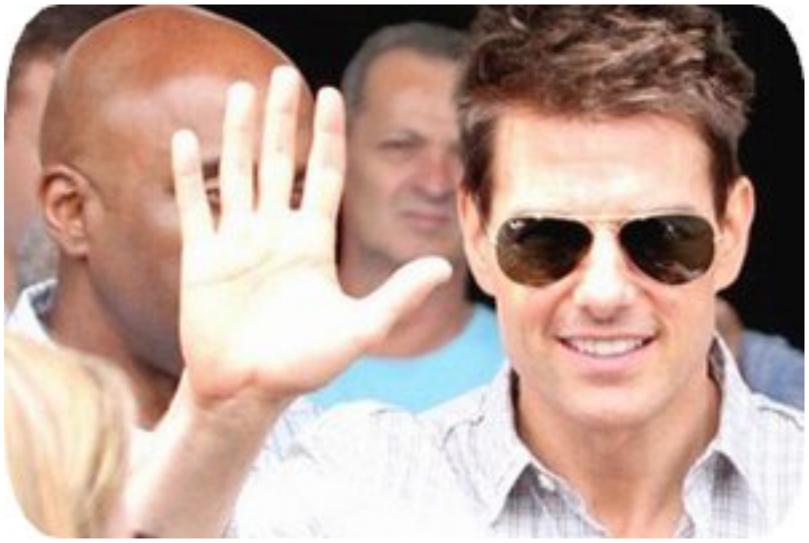
This filter adds rounded corners to the image using the specified color as background.

Arguments

a|b - amount of pixels to use as radius. The argument b is not required, but it specifies the second value for the ellipse used for the radius.

Examples

http://thumbor-server/filters:round_corner(20,255,255,255)/some/image.jpg http://thumbor-server/filters:round_corner(20\T1\textbar{ }40,0,0,0)/some/image.jpg



Sharpen

Usage: sharpen(sharpen_amount,sharpen_radius,luminance_only)

Description

This filter enhances apparent sharpness of the image. It's heavily based on Marco Rossini's excellent Wavelet sharpen GIMP plugin. Check http://registry.gimp.org/node/9836 for details about how it work.

Arguments

- sharpen_amount - Sharpen amount. Typical values are between 0.0 and 10.0.
- sharpen_radius - Sharpen radius. Typical values are between 0.0 and 2.0.
- luminance_only - Sharpen only luminance channel. Values can be 'true' or 'false'

Example 1

http://localhost:8888/unsafe/filters:sharpen(2,1.0,true)/http://videoprocessing.ucsd.edu/~stanleychan/research/pix/Blurred_foreman_0005.png

Example 2

http://localhost:8888/unsafe/filters:sharpen(1.5,0.5,true)/http://images.cambridgeincolour.com/tutorials/sharpening_eagle2-original.jpg

Strip ICC

Usage: strip_icc()

Description

This filter removes any ICC information in the resulting image. Even though the image might be smaller, removing ICC information may result in loss of quality.

Arguments

No arguments

Example

http://localhost:8888/unsafe/filters:strip_icc()/http://videoprocessing.ucsd.edu/~stanleychan/research/pix/Blurred_foreman_0005.png

Upscale

Usage: upscale()

Description

This filter tells thumbor to upscale your images. This only makes sense with "fit-in" or "adaptive-fit-in".

This means that if an original image is 300px width by 200px height and you ask for a 600x500 image, the filter will resize it to 600x400.

Arguments

No arguments allowed.

Example

http://localhost:8888/unsafe/fit-in/600x500/filters:upscale()/https://github.com/thumbor/thumbor/wiki/dice_transparent_background.png

Watermark

Usage: watermark(imageUrl,x,y,alpha)

Description

This filter adds a watermark to the image.

Arguments

- imageUrl - Watermark image URL. It is very important to understand that the same image loader that Thumbor uses will be used here. If this URL contains parentheses they MUST be url encoded, since these are the characters Thumbor uses as delimiters for filter parameters.
- x - Horizontal position that the watermark will be in. Positive numbers indicate position from the left and negative numbers indicate position from the right. If the value is 'center' (without the single quotes), the watermark will be centered horizontally. If the value is 'repeat' (without the single quotes), the watermark will be repeated horizontally.
- y - Vertical position that the watermark will be in. Positive numbers indicate position from the top and negative numbers indicate position from the bottom. If the value is 'center' (without the single quotes), the watermark will be centered vertically. If the value is 'repeat' (without the single quotes), the watermark will be repeated vertically
- alpha - Watermark image transparency. Should be a number between 0 (fully opaque) and 100 (fully transparent).

Example

http://thumbor-server/filters:watermark(http://my.site.com/img.png,-10,-10,50)/some/image.jpg

Crop and Resize Algorithms

thumbor performs the least amount of cropping possible to resize your image to the exact size you specified, without changing it's aspect ratio.

Cropping the image

Before resizing the image, thumbor crops it so it has the same aspect as the desired dimensions. Let's see an example to clarify this concept.

Consider an 800x600 (width x height, in pixels) image and say we want a 400x150 thumbnail of it. The first thing thumbor needs to do is calculate the proportion of the images:

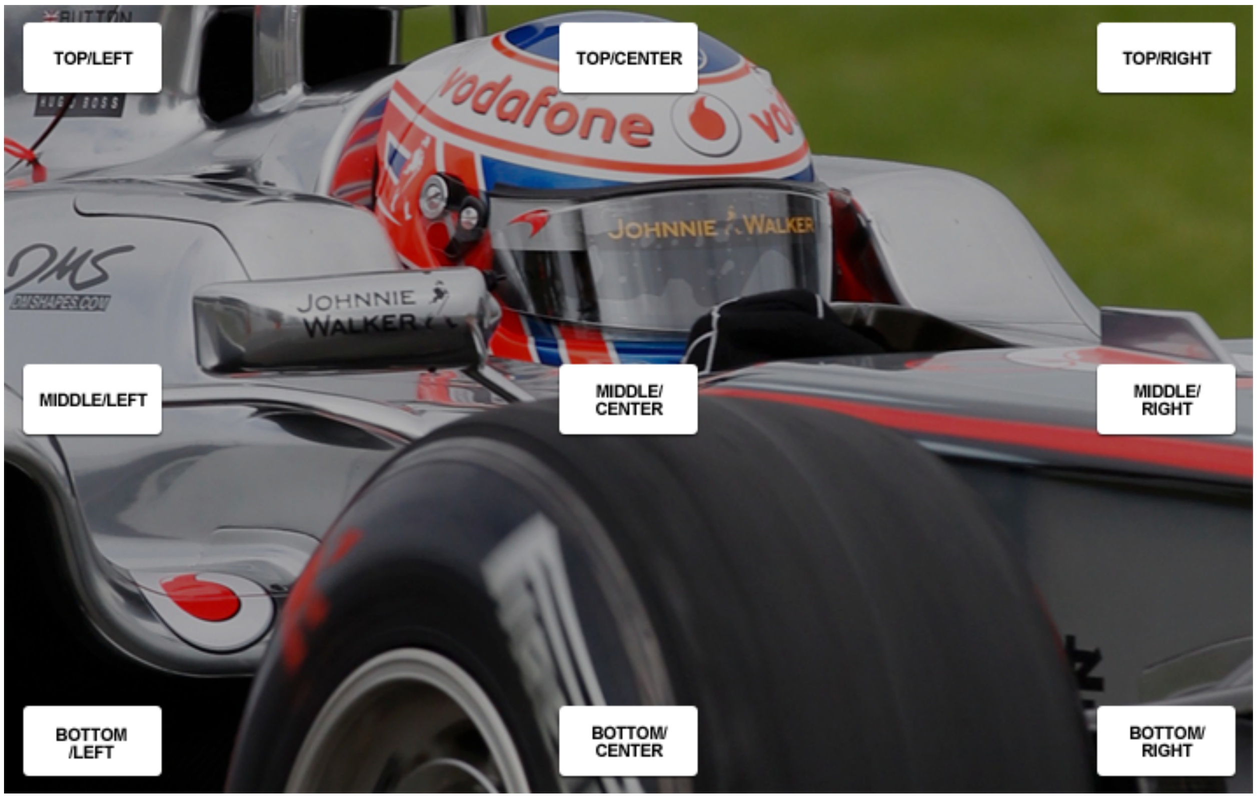
[800 / 600 = 1.333] [400 / 150 = 2.666]

Now that they don't match, thumbor defines if the image needs horizontal or vertical cropping. We never crop both ways, since it's not needed.

So, in our example to get an image of the same proportion of the target one, we need to get the picture height to be 300px (800 is to X as 400 is to 150):

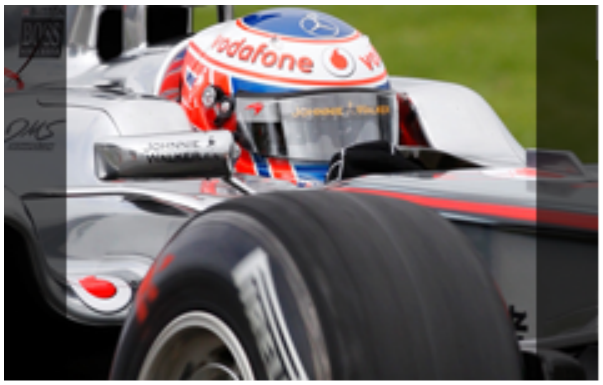
[x = 800 * 150 / 400] [x = 300]

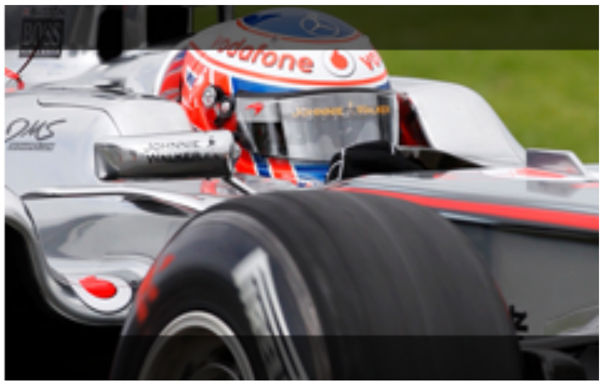
Now all we need to do is cropping 300px of the picture height. To determine whether to crop from the top, bottom or both we use the focal points or the horizontal alignment. If any focal points have been specified we'll use those to find the center of mass of the image (more on that in *Detection Algorithms*). Otherwise we'll use the horizontal and vertical alignments.



Let's say that for this image no focal points were found, so we'll use the vertical alignment to crop the height. Since we specified middle alignment for this example, we'll crop off 150px from the top and 150px from the bottom of the image, similarly to this image:

Here's an example of how thumbor would crop width or height using centered alignment:





Resizing the Image

Now that the image has the same proportion as the image we want, it's just a matter of resizing it to the desired dimensions.

Flipping the Image

If the desired dimensions feature negative numbers, thumbor will flip them around that direction. This means that negative width specifies horizontal flip, while negative height specifies vertical flip.

Detection Algorithms

If the smart mode of thumbor has been specified in the uri (by the /smart portion of it), thumbor will use it's smart detectors to find focal points.

thumbor comes pre-packaged with two focal-point detection algorithms: facial and feature. First it tries to identify faces and if it can't find any, it tries to identify features (more on that below).

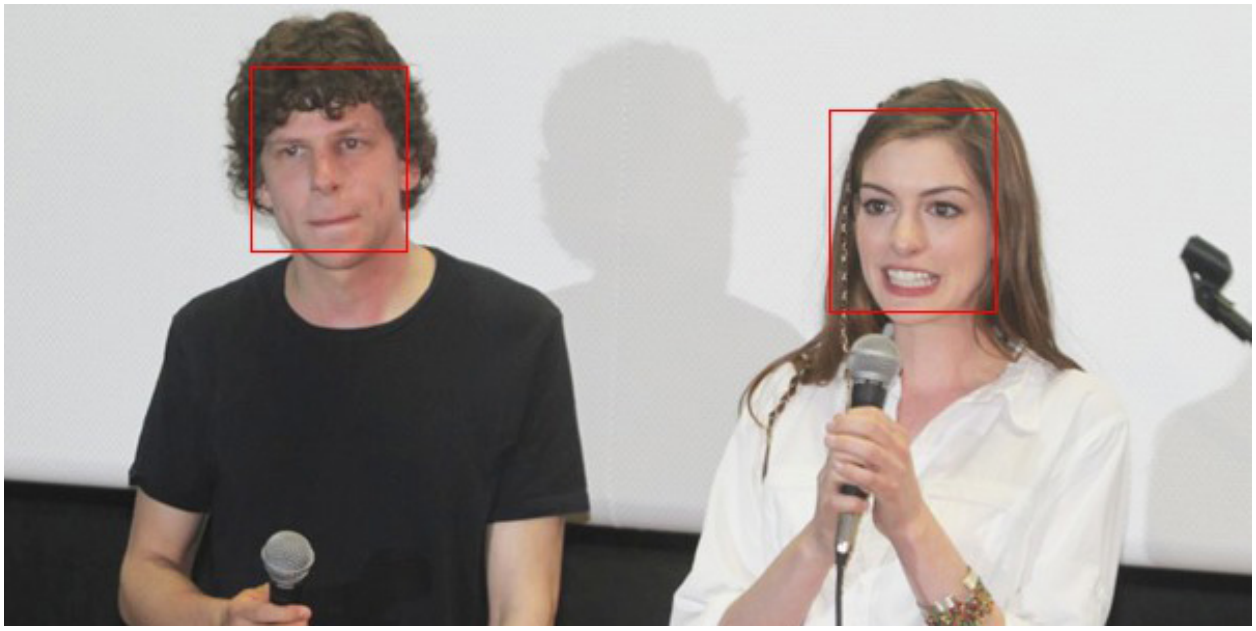
Facial Detection

For instructions on how to get facial detection coordinates see *Metadata Endpoint* .

thumbor uses OpenCV (http://opencv.org) to detect faces. OpenCV returns the rectangle coordinates for the faces it identifies. Thumbor supports changing the file that OpenCV uses for identifying faces. We can see the algorithm in action below:

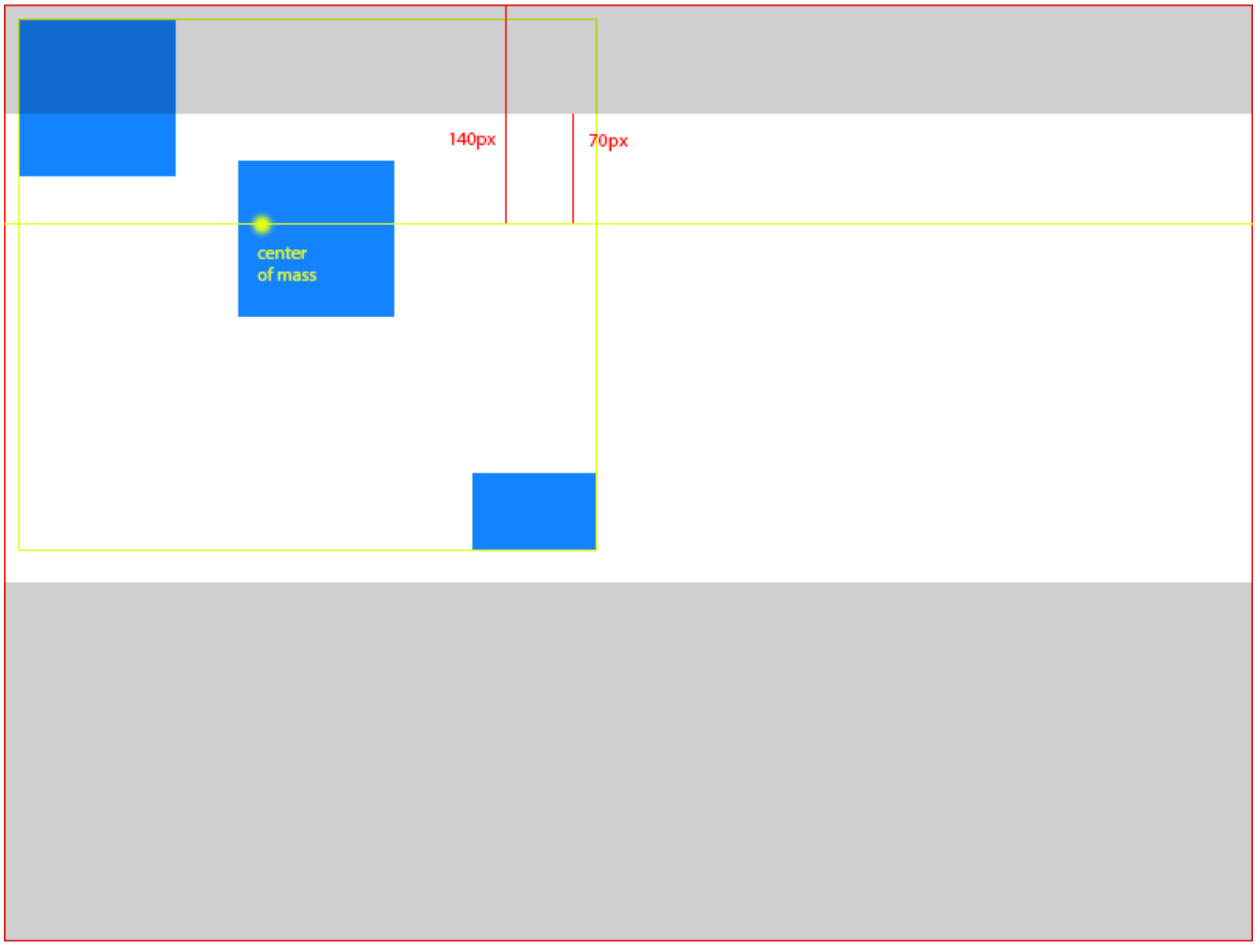
Original image

Red rectangles are the areas identified as faces



After retrieving these squares from OpenCV, thumbor calculates the *center of mass* of the image using *weighted average*.

Consider that OpenCV returned 3 squares at *(10, 10, 100, 100)*, *(150, 100, 100, 100)*, *(300, 300, 80, 50)*, being *(x, y, width, height)*, as such:



In order to find the *center of mass* for all the faces, we must first find the center and weight of each rectangle. We define weight in this scenario as the area of the rectangle.

So, for the faces in our example (***x*,*y*** being the coordinates of the rectangle's center and ***z*** the rectangle weight):

Face 1 - (x = (10 + 100) / 2 = 55), (y = (10 + 100) / 2 = 55), (z = 100 * 100 = 10000)

Face 2 - (x = (150 + 100) / 2 = 125), (y = (100 + 100) / 2 = 100), (z = 100 * 100 = 10000)

Face 3 - (x = (300 + 80) / 2 = 190), (y = (300 + 50) / 2 = 175), (z = 80 * 50 = 4000)

In order to find the *center of mass* we'll do a *weighted average* of the X and Y coordinates of the faces using:

Horizontal Axis - X (((55 * 10000) + (125 * 10000) + (190 * 4000)) / 24000 = 106)

Vertical Axis - Y (((55 * 10000) + (100 * 10000) + (175 * 4000)) / 24000 = 93).

So for the faces found by OpenCV in that image we have the *center of mass* of the picture being *106x93*.

Using Focal Points for Cropping

After finding the center of mass we can use it as the focal point for cropping. Given an image of *800x600* and a focal point at *106x93*, we need to determine the percentage that needs to be cropped from the top, bottom, left and right sides of the image.

To determine the percentage we use simple math to figure how far from the top and the left side the *center of mass* is:

From the left - (106 / 800 * 100 = 13.25%)

From the top - (93 / 600 * 100 = 15.50%)

Using the same example from the *Crop and Resize Algorithms* page, we need to crop *300px* out of the height of the image. In possession of the percentages of crop above, we can calculate how much we need to crop out of the top and bottom with:

Top - (300 * 0.155 ~= 46)

Bottom - just subtract top (*46px*) from the amount of crop (*300px*) - (300 - 46 = 254)

So, now we now we have to remove *46px* out of the top of the picture and *254px* out of the bottom of the picture. In an *800x600* picture, that means cropping from *(0, 46)* to *(800, 346)*, resulting in an *800x300px* image.

Assuming we would crop 300px horizontally, the cropping would be:

Left - (300 * 0.135 ~= 40)

Right - just subtract left (*40px*) from the amount of crop (*300px*) - (300 - 40 = 260)

In an image of *800x600*, that means cropping from *(40, 0)* to *(540, 600)*, resulting in a *500x600px* image. This would not be the case for this image, though.

Feature Detection

If no faces are found in the picture, we still try to find important features in the image, provided by the Good Features to Track Algorithm in OpenCV (http://bit.ly/evAU95).

According to OpenCV documentation, this algorithm finds *"important"* corners in the image. It then returns a list of *(x, y)* values.

We can see the detection taking place in the following images:

The points identified by the good features algorithm:

The cropping based in these features is analogous to the face one, except that all points have a weight of ***1.0*** and are already their centers.

Let's consider that we found **3** feature points: *10x15*, *30x40*, *25x60*. To find the center of mass we would do ((10 + 30 + 25) / 3 ~= 22) to find the horizontal component and ((15 + 40 + 60) / 3 ~= 39) for the vertical one. This means that our center of mass in this scenario is ***22x39***.

Given an image of *800x600* and a center of mass of *22x39*, let's find the left and top percentages:

From the left - (22 / 800 * 100 = 2.75%)

From the top - (93 / 600 * 100 = 6.50%)

Assuming we are cropping *300px* of the height, we'll crop top and bottom according to:

Top - (300 * 0.0275 ~= 9)

Bottom - just subtract top (*9px*) from the amount of crop (*300px*) - (300 - 9 = 291)

In an image of *800x600*, that means cropping from *(0, 9)* to *(800, 309)*, resulting in a *800x300px* image.

If we were cropping *300px* of the width instead, we would crop left and right according to:

Left - (300 * 0.065 ~= 20)

Right - just subtract left (*20px*) from the amount of crop (*300px*) - (300 - 20 = 280)

In an image of *800x600*, that means cropping from *(20, 0)* to *(520, 600)*, resulting in a *500x600px* image.

How to upload Images

Thumbor provides a `/image` REST end-point to upload your images and manage it.

This way you can send thumbor your original images by doing a simple post to its urls.

Configuration

The table below show all configuration parameters to manage image upload :

Configuration parameter	Default	Description
UPLOAD_ENABLED	False	Indicates whether thumbor should enable File uploads
UPLOAD_PUT_ALLOWED	False	Indicates whether image overwrite should be allowed
UP-LOAD_DELETE_ALLOWED	False	Indicates whether image deletion should be allowed
UPLOAD_PHOTO_STORAGE	thum-bor.storages.file_storage	The type of storage to store uploaded images with
UP-LOAD_DEFAULT_FILENAME	image	Default filename for image uploaded
UPLOAD_MAX_SIZE	0	Max size in Kb for images uploaded to thumbor
MIN_WIDTH	1	Min width in pixels for images uploaded
MIN_HEIGHT	1	Min width in pixels for images uploaded

Thumbor comes with the `/image` REST end-point to upload disabled by default. In order to enable it, just set the `UPLOAD_ENABLED` configuration in your thumbor.conf file to `True`.

Thumbor will then use the storage specified in the `UPLOAD_PHOTO_STORAGE` configuration to save your images. You can use an existing storage (filesytem, redis, mongo, hbase...) or *create your own storage* if needed .

You can manage image putting and deletions simply set the configuration parameters `UPLOAD_PUT_ALLOWED` and `UPLOAD_DELETE_ALLOWED` to `True`. This parameters are set to `False` by default for security reasons.

Finally the upload constraints (max size, image width and height) will be controlled by `UPLOAD_MAX_SIZE`, `MIN_WIDTH` and `MIN_HEIGHT` parameters.

API Usage

The Thumbor `/image` REST end-point supports the commons HTTP methods : * POST : to upload a new image * GET : to display an image uploaded * PUT : to replace an image uploaded by another preserving the URI * DELETE : to remove an image uploded from storage

By default, `PUT` and `DELETE` methods are disabled as explained above. This is done to tighten thumbor's security.

Posting

Posting is the only allowed by default method when you activate the upload module. It allows new images to be sent to Thumbor.

In order to upload a new image, you have two choices : * send an HTTP **POST** to the `/image` end-point with the image as payload (REST style) * send an HTTP **POST** to the `/image` end-point with a multi-part form with a file field called media (Form style).

In the REST style mode you may add an optional `Slug` header to define the image filename, which is useful for SEO reasons. Not specifying a `Slug` causes the server to use the default filename for the image (`UPLOAD_DEFAULT_FILENAME` parameter) .

The HTTP response will return a `Location` header pointing on the uploaded image. The URI presents in `Location` header may be used to update or delete the image uploaded (see below).

HTTP status code

The status code returned will be :

- 201 Created (success)
- 415 Unsupported Media Type (image type is not allowed)
- 412 Precondition Failed (image is too small or the file is not an image)

Putting

Putting is a little more dangerous if you don't have strict control of who can access the `/image` end-point. This is because whatever is sent using this method gets saved to storage, overwriting the previous entry.

In order to replace an existing image, all you have to do is send an HTTP **PUT** request to the `/image` end-point with the new image content as payload. The new image will replace the original image preserving the URI.

As for the `POST` method you may add an optional `Slug` header to define the image filename.

The HTTP response will return a `Location` header pointing on the modified image. The URI presents in `Location` header may be used to update again the image or delete it.

HTTP status code

The status code returned will be :

- 204 No Content (success)
- 405 Method Not Allowed (if thumbor's configuration disallows putting images)
- 415 Unsupported Media Type (image type is not allowed)
- 412 Precondition Failed (image is too small or file is not an image)

Deleting

Deleting can be very dangerous, thus is disabled by default.

If you do enable it, in order to delete an image, all you have to do is send an HTTP **DELETE** request to the `/image` end-point.

HTTP status code

- 204 No Content (success)
- 404 Not Found (image doesn't exists)
- 405 Method Not Allowed (if thumbor's configuration disallows deleting images)

Example :

Assuming the thumbor server is located at : `http://thumbor-server`

Upload an image via the REST API

When using the `/image` REST end-point to upload your image via the REST API :

```
curl -i -H "Content-Type: image/jpeg" -H "Slug: photo.jpg"
        -XPOST http://thumbor-server/image --data-binary "@tests/fixtures/images/
↪20x20.jpg"
```

the HTTP **POST** request was send to the server :

```
POST /image
Content-Type: image/jpeg
Content-Length: 822
Slug : photo.jpg
```

and the Thumbor server should return :

```
HTTP/1.1 201 Created
Content-Length: 0
Content-Type: text/html; charset=UTF-8
Location: /image/05b2eda857314e559630c6f3334d818d/photo.jpg
Server: TornadoServer/2.1.1
```

The image is created at `http://thumbor-server/image/05b2eda857314e559630c6f3334d818d/photo.jpg`. It can be retrieved, modified or deleted via this URI.

The optional `Slug` HTTP header specifies the filename to use for the image uploaded.

Upload an image via a form

When using the `/image` REST end-point to upload your images via a form, the user is free to choose the filename of the image via the `filename` field :

```
curl -i -XPOST http://thumbor-server/image
        -F "media=@tests/fixtures/images/20x20.jpg;type=image/jpeg;filename=croco.jpg"
```

the HTTP **POST** request was send to the server :

```
POST /image
Content-Type: multipart/form-data; boundary=-----------------------------11df125d8b12
Content-Length: 822
```

and the Thumbor server should return :

```
HTTP/1.1 201 Created
Content-Length: 0
Content-Type: text/html; charset=UTF-8
Location: /image/05b2eda857314e559630c6f3334d818d/croco.jpg
```

The image is created at `http://thumbor-server/image/05b2eda857314e559630c6f3334d818d/croco.jpg`. It can be retrieve, modify or delete via this URI using the REST API.

Modifying an image

To replace the previously uploaded image by another we use:

```
curl -i -H "Content-Type: image/jpeg" -H "Slug: modified_image.jpg"
        -XPUT http://thumbor-server/image/05b2eda857314e559630c6f3334d818d/photo.jpg -
↪-data-binary "@tests/fixtures/images/20x20.jpg"
```

the HTTP **PUT** request was send to the server :

```
PUT /image/05b2eda857314e559630c6f3334d818d/photo.jpg
Content-Type: image/jpeg
Content-Length: 822
Slug : modified_image.jpg
```

and the Thumbor server should return :

```
HTTP/1.1 204 No Content
Content-Length: 0
Content-Type: text/html; charset=UTF-8
Location: /image/05b2eda857314e559630c6f3334d818d/modified_image.jpg
Server: TornadoServer/2.1.1
```

Deleting an image

Finally to delete the uploaded image we use:

```
curl -i -XDELETE http://thumbor-server/image/05b2eda857314e559630c6f3334d818d/
↪modified_image.jpg
```

the HTTP **DELETE** request was send to the server :

```
DELETE /image/05b2eda857314e559630c6f3334d818d/modified_image.jpg
```

and the Thumbor server should return :

```
HTTP/1.1 204 No Content
Content-Length: 0
Content-Type: text/html; charset=UTF-8
Server: TornadoServer/2.1.1
```

Image loader

Pre-packaged loaders

thumbor comes pre-packaged with http and filesystem loaders.

Http loader

The http loader gets the original image portion of the URI and performs an HTTP GET to it. It then returns the image's string representation.

The http loader uses the **ALLOWED_SOURCES** configuration to determine whether or not an image is from a trusted source and can thus be loaded.

You can specify the maximum size of the source image to be loaded. The http loader first gets the image size (without loading its contents), checks against your specified size and returns 404 if the source image size is larger than the max size. The max size option is **MAX_SOURCE_SIZE** and the default is no maximum size.

To use it you should set the **LOADER** configuration to **'thumbor.loaders.http_loader'**.

Https loader

The https loader works the same way as the http loader, except that it defaults to https instead of http.

To use it you should set the **LOADER** configuration to **'thumbor.loaders.https_loader'**.

Strict https loader

The strict https loader works the same way as the http loader, except that it only allows to load images over https.

To use it you should set the **LOADER** configuration to **'thumbor.loaders.strict_https_loader'**.

File loader

The file loader gets the original image portion of the URI and retrieves the file from the file system from a known path specified by the **FILE_LOADER_ROOT_PATH** configuration.

It joins the specified path with the configured root path and reads the image file if it exists.

To use it you should set the **LOADER** configuration to **'thumbor.loaders.file_loader'**.

File loader with http loader fallback

In some environments you need both kinds of file loading. For this use case you can use as loader with built-in fallback.

This loader will try to load images from local file storage. In case of an error the loader retry to load image with http_loader. If both attempts failed you'll get an error.

To use it you should set the **LOADER** configuration to **'thumbor.loaders.file_loader_http_fallback'**.

Custom loaders

If thumbor image loaders do not meet your needs you can implement a new image loader.

The structure of the module you should implement can be seen in the http loader at https://github.com/thumbor/thumbor/blob/master/thumbor/loaders/http_loader.py.

The only required method to implement is the one that receives the portion of the URI that has the original image path, named **load**. This method also receives a callback and should call the callback with the results of reading the image.

Another example can be seen in the filesystem loader at https://github.com/thumbor/thumbor/blob/master/thumbor/loaders/file_loader.py.

You can optionally implement a validate(URI) method that thumbor will call to make sure that your loader can accept the user required URI.

Image storage

thumbor uses image storages to perform less retrievals of images from the sources, thus potentially saving expensive resources (such as outbound network).

Pre-Packaged Storages

thumbor comes with **filesystem**, **redis** and **mongodb** storages. There's also a **nostorage** storage for debugging or benchmarking purposes.

Filesystem Storage

thumbor can store original images in the filesystem.

The file storage uses the **FILE_STORAGE_ROOT_PATH** configuration to save the images. It then joins the original image part of the URI to create the proper path to store the image in the filesystem.

There's a **STORAGE_EXPIRATION_SECONDS** option that will determine the time in seconds that a file is considered to be expired. When a file is expired, thumbor will try to retrieve the file using the specified *Image loader*.

To use the filesystem storage set the configuration option of **STORAGE** to **'thumbor.storages.file_storage'**.

Redis Storage

Redis eliminates the risks of locks as well. In order to use the redis storage set the configuration option of **STORAGE** to **'thumbor.storages.redis_storage'**.

You also need to configure the redis connection information using the **REDIS_STORAGE_SERVER_PORT**, **REDIS_STORAGE_SERVER_HOST** and **REDIS_STORAGE_SERVER_DB**, like this:

```
REDIS_STORAGE_SERVER_HOST = 'localhost'
REDIS_STORAGE_SERVER_PORT = 6379
REDIS_STORAGE_SERVER_DB = 0
```

There's a **STORAGE_EXPIRATION_SECONDS** option that will determine the time in seconds that a file is considered to be expired. When a file is expired, thumbor will try to retrieve the file using the specified *Image loader*.

MongoDB Storage

In order to use the MongoDB storage set the configuration option of **STORAGE** to **'thumbor.storages.mongo_storage'**.

You also need to configure the mongo connection information using the **MONGO_STORAGE_SERVER_PORT**, **MONGO_STORAGE_SERVER_HOST**, **MONGO_STORAGE_SERVER_DB** and **MONGO_STORAGE_SERVER_COLLECTION**, like this:

```
MONGO_STORAGE_SERVER_HOST = 'localhost'
MONGO_STORAGE_SERVER_PORT = 27017
MONGO_STORAGE_SERVER_DB = 0
MONGO_STORAGE_SERVER_COLLECTION = 'images'
```

There's a **STORAGE_EXPIRATION_SECONDS** option that will determine the time in seconds that a file is considered to be expired. When a file is expired, thumbor will try to retrieve the file using the specified *Image loader*.

NoStorage Storage

This is a storage intended for debugging or benchmarking purposes. It does not store any images and always returns None when thumbor asks for an image.

In order to use this storage set the configuration option of **STORAGE** to **'thumbor.storages.no_storage'**.

MixedStorage Storage

This is a storage intended for scenarios where you want to store the original images files one way and the security key another (or detector information).

A good example would be storing files in the filesystem, while storing security keys in a database.

In order to use this storage set the configuration option of **STORAGE** to **'thumbor.storages.mixed_storage'**.

You must specify the `MIXED_STORAGE_FILE_STORAGE`, `MIXED_STORAGE_CRYPTO_STORAGE` and `MIXED_STORAGE_DETECTOR_STORAGE` options to define the original images storage, the security key storage and the detector results storage, respectively. Here's a sample configuration:

```
MIXED_STORAGE_FILE_STORAGE = 'thumbor.storages.file_storage'
MIXED_STORAGE_CRYPTO_STORAGE = 'thumbor.storages.redis_storage'
MIXED_STORAGE_DETECTOR_STORAGE = 'thumbor.storages.redis_storage'

FILE_STORAGE_ROOT_PATH = '/tmp/mypath'

REDIS_STORAGE_SERVER_HOST = 'localhost'
REDIS_STORAGE_SERVER_PORT = 6379
REDIS_STORAGE_SERVER_DB = 0
```

As you can see, you still have to tell thumbor the specific configurations for each storage you choose.

Custom Storages

If the built-in storages do not suit your needs, you can always implement your own storage and use it in the **STORAGE** configuration.

All you have to do is create a class called Storage that inherits from BaseStorage in your module, as can be seen in https://github.com/thumbor/thumbor/blob/master/thumbor/storages/file_storage.py.

Lazy Detection

Rationale

Thumbor performs pipeline detection of focal points for a given image. What this means is that it tries to determine one detection at a time, only skipping to the next if the current one fails.

We could configure it to run frontal face detection, then if it fails, try profile face detection and if it fails, best features detection.

As you can imagine, this is a cumbersome process and can take up precious cpu time from your server(s), eventually leading it to starvation of CPU. This is why we've implemented what we call Queued Detection.

Queued Detection

Configuring thumbor for lazy detecting is as simple as specifying a detector that supports queued detection.

Thumbor ships with three such detectors, called:

- thumbor.detectors.queued_detector.queued_complete_detector
- thumbor.detectors.queued_detector.queued_face_detector
- thumbor.detectors.queued_detector.queued_feature_detector

These are responsible, respectively, for pipeline detection of face and feature, only face or only feature.

You can check what additional configuration you need to add to your configuration file (thumbor.conf) in order to have the bundled detectors working.

How thumbor deals with queued detection?

When an image request arrives with a flag of "smart" detection, a call is made to the queued detector and it tells thumbor to skip smart detection and to serve the image with non-smart cropping (much faster).

The call to the queued detector places a message in a Redis Queue that will later be processed in order to detect focal points in the image.

The next time a request arrive for the same image and with a flag of "smart" detection, if information on detection is already available (if the message in the queue has already been processed), thumbor uses that info to do smart cropping and serves the result.

If the image still hasn't been processed, the same process from before applies, except thumbor won't place another message in the queue. This is intended as a way not to flood the queue with requests for the same image.

Posting, Putting and Deleting

thumbor original photo uploading end-point supports three different http verbs: put, post and delete.

By default, put and delete are disabled. This is done to tighten thumbor's security. If you wish to enable them, please refer to the *How to upload Images* page.

Posting

Posting is the only allowed by default method. It allows new images to be sent to thumbor. If the same image is sent again, thumbor will raise an exception.

This is done so users can't overwrite images with other images, possibly defacing your website.

In order to post a new image, all you have to do is send a multi-part form with a file field called media and action of `http://{thumbor-server}/image` and method of `POST`.

HTTP status code

- 201 Created (success)
- 409 Conflict (image already exists)
- 412 Precondition Failed (image is too small or the file is not an image)

Putting

Putting is a little more dangerous if you don't have strict control of who can access the `/image` route. This is because whatever is sent using this method gets saved to storage, overwriting the previous entry.

In order to put a new image, all you have to do is send a multi-part form with a file field called `media` and action of `http://{thumbor-server}/image` and method of `PUT`.

HTTP status code

- 201 Created (success)
- 405 Method Not Allowed (if thumbor's configuration disallows putting images)
- 412 Precondition Failed (image is too small or file is not an image)

Deleting

Deleting can be very dangerous, thus is disabled by default.

If you do enable it, in order to delete an image, all you have to do is send a request to `http://{thumbor-server}/image` with a method of `DELETE` and a field called `file_path` with the same path that was used when uploading the image.

HTTP status code

- 200 OK (success)
- 405 Method Not Allowed (if thumbor's configuration disallows deleting images)

Result Storage

thumbor uses a result storage to improve the speed of responding subsequent requests for the same image.

When a request for a given image with a set of parameters arrive, thumbor processes the request and before returning it, asks for the result storage to store it.

The next time the same request arrives, it will get it from the result storage and return it, thus saving a lot of processing.

Pre-packaged result storages

thumbor comes pre-packaged with a filesystem result storage.

Filesystem

The file system result storage, as the name implies, stores images in the filesystem.

Images are stored in whatever path is specified in the `RESULT_STORAGE_FILE_STORAGE_ROOT_PATH`, and consequently retrieved from the same path.

By default, the file system result storage keeps images forever. You are allowed to specify an expiration, though, using the `RESULT_STORAGE_EXPIRATION_SECONDS` configuration. Again, as the name implies, it specifies the number of seconds with which files expire.

To use it you should set the RESULT_STORAGE configuration to 'thumbor.result_storages.file_storage'.

Creating a custom result storage

In order to implement your own result storage, you have to implement a few methods. A reference implementation can be found at the File Storage.

The required methods are put, get, validate_path and normalize_path.

Optimizers

jpegtran

Jpegtran is a lossless jpeg optimizer which can make your jpegs smaller by optimizing DCT coefficients. Information on jpegtran can be a bit difficult to find but the linux man page is pretty good: https://linux.die.net/man/1/jpegtran

Jpegtran can be used in conjunction with Thumbor. If the optimizer has been activated, Thumbor will first process your jpeg normally then it will hand the jpeg off to jpegtran for further optimizations before Thumbor returns the final image.

To use jpegtran with Thumbor you must first install jpegtran, various linux distros often provide a package by the same name or it can be installed from source. You should make sure that jpegtran is in PATH, do a *which jpegtran* and you should see an absolute path where the jpegtran resides.

You also need to enable the Thumbor jpegtran optimizer in your thumbor.conf, like so:

```
OPTIMIZERS = [
  'thumbor.optimizers.jpegtran'
]
```

You can also manually specify the jpegtran path, like this:

```
JPEGTRAN_PATH=/usr/local/bin/jpegtran
```

Once activated, no extra url parameters are needed - jpegtran will run on all jpegs automatically. If you have opted to use progressive jpegs via the PROGRESSIVE_JPEG option, jpegtran will also honor and product progressive jpegs.

gifv

The gifv optimizer is able to convert gifs to mp4 or webm videos, often resulting in dramatically smaller sized files.

Gifv is categorized as experimental and should be used with caution. It uses ffmpeg to convert gifs to videos and so it's sensitive to changes with ffmpeg. It's recommended to lock your ffmpeg version with a fixed version (chef, docker, etc) and if updating make sure to check that the update doesn't break gifv. **FFmpeg version 3.2.4 is the current recommended version.** Later version, such as 3.3 will break the proper conversion of gif delays to frame durations in videos ... meaning videos will not be the same length as equivelant gifs.

To enable gifv, ensure ffmpeg is in PATH and enable the optimizer in your config:

```
OPTIMIZERS = [
  'thumbor.optimizers.gifv',
]
```

Once activated, you must add the gifv() option to your filters list. An example request might look like this:

```
http://localhost:8888/unsafe/filters:gifv()/http://localhost/livingroom.gif
```

The above example will default to using the mp4 video container with h264 video. You can also be explicit:

```
http://localhost:8888/unsafe/filters:gifv(mp4)/http://localhost/livingroom.gif
```

or use explicitly specify webm

```
http://localhost:8888/unsafe/filters:gifv(webm)/http://localhost/livingroom.gif
```

Because videos (in mp4 or webm format) cannot contain alpha transparency a background color will be automatically added. The default color is white. You can also specify a background color:

```
http://localhost:8888/unsafe/filters:gifv():background_color(ff00ff)/http://localhost/
↪livingroom.gif
```

```
http://localhost:8888/unsafe/filters:gifv():background_color(f0f)/http://localhost/
↪livingroom.gif
```

```
http://localhost:8888/unsafe/filters:gifv():background_color(magenta)/http://
↪localhost/livingroom.gif
```

The color must be specified in 6 character hex, 3 character hex or color name. But 6 or 3 character hex are the preferred formats. Including a # symbol in your color will break the url if not url encoded and thumbor will error on the request. The recommendation is to not use them at all which also makes urls shorter. But if you must a leading *%23* will probably work.

Administration

Configuration

thumbor's configuration file is just a regular python script that thumbor loads.

In order to get a commented configuration file, just run:

```
thumbor-config > ./thumbor.conf
```

Extensibility Section

LOADER

The loader is responsible for retrieving the source image that thumbor will work with. This configuration defines the module that thumbor will use for it. **This must be a full namespace module (a.k.a. python has to be able to **import** it).**

i.e.: thumbor.loaders.http_loader

STORAGE

The storage is responsible for storing the source image bytes and related metadata (face-detection, encryption and such) so that we don't keep loading it every time. **This must be a full namespace module (a.k.a. python has to be**

able to **import it).**

i.e.: thumbor.storages.file_storage

MIXED_STORAGE_FILE_STORAGE

If you are using thumbor's mixed storage (thumbor.storages.mixed_storage), this is where you specify the storage that will be used to store images. **This must be a full namespace module (a.k.a. python has to be able to **import** it).**

i.e.: thumbor.storages.file_storage

MIXED_STORAGE_CRYPTO_STORAGE

If you are using thumbor's mixed storage (thumbor.storages.mixed_storage), this is where you specify the storage that will be used to store cryptography information. **This must be a full namespace module (a.k.a. python has to be able to **import** it).**

i.e.: thumbor.storages.redis_storage

MIXED_STORAGE_DETECTOR_STORAGE

If you are using thumbor's mixed storage (thumbor.storages.mixed_storage), this is where you specify the storage that will be used to store facial and feature detection results. **This must be a full namespace module (a.k.a. python has to be able to **import** it).**

i.e.: thumbor.storages.mongo_storage

ENGINE

The engine is responsible for transforming the image. **This must be a full namespace module (a.k.a. python has to be able to **import** it).**

Thumbor ships with three imaging engines:

- thumbor.engines.pil
- thumbor.engines.opencv
- thumbor.engines.graphicsmagick
- thumbor.engines.imagemagick

  (This engine isn't supported anymore, check #52)

RESULT_STORAGE

The result storage is responsible for storing the resulting image with the specified parameters (think of it as a cache), so that we don't keep processing it every time a request comes in. **This must be a full namespace module (a.k.a. python has to be able to **import** it).**

i.e.: thumbor.result_storages.file_storage

URL_SIGNER

The url signer is responsible for validation and signing of requests to prevent url tampering, which could lead to denial of service (example: filling the result_storage by specifying a different size). **This must be a full namespace module (a.k.a. python has to be able to **import** it).**

i.e.: libthumbor.url_signers.base64_hmac_sha1

Filters Section

In order to specify the filters that thumbor will use, you need a configuration key called FILTERS. This is a regular python list with the full names (names that python can import) of the filter modules you want to use.

An example:

```
FILTERS = [
    'thumbor.filters.brightness',
    'thumbor.filters.contrast',
    'thumbor.filters.rgb',
    'thumbor.filters.round_corner',
    'thumbor.filters.quality',
    'thumbor.filters.noise',
    'thumbor.filters.watermark',
]
```

Metadata Section

META_CALLBACK_NAME

If you want thumbor to use JSONP for image metadata instead of using JSON, just set this variable to the callback name you want.

i.e.: “thumbor_callback”

Face and Feature Detection Section

DETECTORS

This options specifies the detectors that should run the image to check for focal points.

i.e.: [”thumbor.detectors.face_detector”, “thumbor.detectors.feature_detector”]

FACE_DETECTOR_CASCADE_FILE

This option specifies the cascade (XML) file path to train openCV to find faces.

i.e.: haarcascade_frontalface_alt.xml

Imaging Section

ALLOWED_SOURCES

This configuration defines the source of the images that thumbor will load. This is only used in the HttpLoader (check the LOADER configuration above).

i.e.: ALLOWED_SOURCES=['http://s.glbimg.com']

Another example with wildcards:

ALLOWED_SOURCES = ['.+.globo.com', '.+.glbimg.com']

This is to get any images that are in *.globo.com or *.glbimg.com and it will fail with any other domains.

MAX_WIDTH and MAX_HEIGHT

These define the box that the resulting image for thumbor must fit-in. This means that no image that thumbor generates will have a width larger than MAX_WIDTH or height larger than MAX_HEIGHT.

i.e.:

```
MAX_WIDTH = 1200
MAX_HEIGHT = 800
```

MIN_WIDTH and MIN_HEIGHT

These define the box that the resulting image for thumbor must fit-in. This means that no image that thumbor generates will have a width smaller than MIN_WIDTH or height smaller than MIN_HEIGHT.

i.e.:

```
MIN_WIDTH = 1
MIN_HEIGHT = 1
```

QUALITY

This option defines the quality that JPEG images will be generated with. It defaults to 80.

i.e.: QUALITY = 90

MAX_AGE

This option defines the number of seconds that images should remain in the browser's cache. It relates directly with the Expires and Cache-Control headers.

i.e.: MAX_AGE = 24 * 60 * 60 # A day of caching

MAX_AGE_TEMP_IMAGE

When an image has some error in its detection or it has deferred queueing, it's convenient to set a much lower expiration time for the image cache. This way the browser will request the proper image faster.

This option defines the number of seconds that images in this scenario should remain in the browser's cache. It relates directly with the Expires and Cache-Control headers.

i.e.: MAX_AGE_TEMP_IMAGE = 60 # A minute of caching

RESPECT_ORIENTATION

If this option is set to True, thumbor will reorient the image according to it's EXIF Orientation tag (if one can be found). This options defaults to False.

The operations performed in the image are as follow (considering the value of the Orientation EXIF tag):

1. Nothing
2. Flips the image horizontally
3. Rotates the image 180 degrees
4. Flips the image vertically
5. Flips the image vertically and rotates 270 degrees
6. Rotates the image 270 degrees
7. Flips the image horizontally and rotates 270 degrees
8. Rotates the image 90 degrees

i.e.: RESPECT_ORIENTATION = False

ALLOW_ANIMATED_GIFS

This option indicates whether animated gifs should be supported.

i.e.: `ALLOW_ANIMATED_GIFS = True`

USE_GIFSICLE_ENGINE

This option indicates whether gifsicle should be used for all gif images, instead of the actual imaging engine. This defaults to False.

When using gifsicle thumbor will generate proper animated gifs, as well as static gifs with the smallest possible size.

i.e.: `USE_GIFSICLE_ENGINE = True`

WARNING: When using gifsicle engine, filters will be skipped. Thumbor will not do smart cropping as well.

AUTO_WEBP

This option indicates whether thumbor should send WebP images automatically if the request comes with an "Accept" header that specifies that the browser supports "image/webp".

i.e.: `AUTO_WEBP = True`

Queueing - Redis

REDIS_QUEUE_SERVER_HOST

Server host for the queued redis detector.

i.e.: `REDIS_QUEUE_SERVER_HOST = 'localhost'`

REDIS_QUEUE_SERVER_PORT

Server port for the queued redis detector.

i.e.: `REDIS_QUEUE_SERVER_PORT = 6379`

REDIS_QUEUE_SERVER_DB

Server database index for the queued redis detector

i.e.: `REDIS_QUEUE_SERVER_DB = 0`

REDIS_QUEUE_SERVER_PASSWORD

Server password for the queued redis detector

i.e.: `REDIS_QUEUE_SERVER_PASSWORD = None`

Queueing - Amazon SQS

SQS_QUEUE_KEY_ID

Amazon AWS key id.

i.e.: `SQS_QUEUE_KEY_ID = None`

SQS_QUEUE_KEY_SECRET

Amazon AWS key secret.

i.e.: `SQS_QUEUE_KEY_SECRET = None`

SQS_QUEUE_REGION

Amazon AWS SQS region.

i.e.: `SQS_QUEUE_REGION = 'us-east-1'`

Security Section

SECURITY_KEY

This option specifies the security key that thumbor uses to sign secure URLs.

i.e.: 1234567890123456

ALLOW_UNSAFE_URL

This option specifies that the /unsafe url should be available in this thumbor instance. It is boolean (True or False).

ALLOW_OLD_URLS

This option specifies that the format prior to 3.0.0 release changes should be allowed. It defaults to True.

THIS OPTION IS DEPRECATED AND WILL DEFAULT TO FALSE IN THE NEXT MAJOR.

Loader Options Section

FILE_LOADER_ROOT_PATH

In case you are using thumbor's built-in file loader, this is the option that allows you to specify where to find the images.

HTTP_LOADER_DEFAULT_USER_AGENT

This option allows users to specify the default user-agent that thumbor will send when requesting images with the HTTP Loader. Defaults to 'Thumbor/' (like Thumbor/3.10.0).

HTTP_LOADER_FORWARD_USER_AGENT

This option tells thumbor to forward the request user agent when requesting images using the HTTP Loader. Defaults to False.

Storage Options Section

STORAGE_EXPIRATION_SECONDS

This options specifies the default expiration time in seconds for the storage.

i.e.: 60 (1 minute)

STORES_CRYPTO_KEY_FOR_EACH_IMAGE

This option specifies whether thumbor should store the key for each image (thus allowing the image to be found even if the security key changes). This is a boolean flag (True or False).

File Storage Section

FILE_STORAGE_ROOT_PATH

In case you are using thumbor's built-in file storage, this is the option that allows you to specify where to save the images.

MongoDB Storage Section

MONGO_STORAGE_SERVER_HOST

This is the option that specifies the host for mongodb.

i.e.: 127.0.0.1

MONGO_STORAGE_SERVER_PORT

This is the option that specifies the port where mongodb is running in.

i.e.: 27017

MONGO_STORAGE_SERVER_DB

This is the option that specifies the database for mongodb.

i.e.: thumbor

MONGO_STORAGE_SERVER_COLLECTION

This is the option that specifies the collection for thumbor's documents.

i.e.: images

Redis Storage Section

REDIS_STORAGE_SERVER_HOST

This option specifies the host server for Redis.

i.e.: localhost

REDIS_STORAGE_SERVER_PORT

This option specifies the port that redis is listening in.

i.e.: 6379

REDIS_STORAGE_SERVER_DB

This option specifies the database that thumbor should use.

i.e.: 0

REDIS_STORAGE_SERVER_PASSWORD

This option specifies the password that thumbor should use to authenticate with redis.

i.e.: my-redis-password

Memcached Storage Section

MEMCACHE_STORAGE_SERVERS

List of Memcache storage server hosts.

i.e.: `MEMCACHE_STORAGE_SERVERS = ['localhost:11211']`

Result Storage Section

RESULT_STORAGE_EXPIRATION_SECONDS

Expiration in seconds of generated images in the result storage.

i.e.: `RESULT_STORAGE_EXPIRATION_SECONDS = 0`

RESULT_STORAGE_FILE_STORAGE_ROOT_PATH

Path where the Result storage will store generated images.

i.e.: `RESULT_STORAGE_FILE_STORAGE_ROOT_PATH = '/tmp/thumbor/result_storage'`

RESULT_STORAGE_STORES_UNSAFE

Indicates whether unsafe requests should also be stored in the Result Storage.

i.e.: `RESULT_STORAGE_STORES_UNSAFE = False`

Logging

THUMBOR_LOG_FORMAT

This option specifies the format to be used by logging messages sent from thumbor.

i.e.: '%(asctime)s %(name)s:%(levelname)s %(message)s'

THUMBOR_LOG_DATE_FORMAT

This option specifies the date format to be used by logging messages sent from thumbor.

i.e.: '%Y-%m-%d %H:%M:%S'

Error Handling

USE_CUSTOM_ERROR_HANDLING

This configuration indicates whether thumbor should use a custom error handler.

i.e.: `USE_CUSTOM_ERROR_HANDLING = False`

ERROR_HANDLER_MODULE

Error reporting module. Needs to contain a class called ErrorHandler with a handle_error(context, handler, exception) method.

i.e.: `ERROR_HANDLER_MODULE = 'thumbor.error_handlers.sentry'`

Error Handling - Sentry

SENTRY_DSN_URL

Sentry thumbor project dsn. i.e.: http://5a63d58ae7b94f1dab3dee740b301d6a:73eea45d3e8649239a973087e8f21f98@localhost:9000/2

i.e.: `SENTRY_DSN_URL = ''`

Upload

UPLOAD_MAX_SIZE

Max size in Kb for images uploaded to thumbor.

i.e.: `UPLOAD_MAX_SIZE = 0`

UPLOAD_ENABLED

Indicates whether thumbor should enable File uploads.

i.e.: `UPLOAD_ENABLED = False`

UPLOAD_PHOTO_STORAGE

The type of storage to store uploaded images with.

i.e.: `UPLOAD_PHOTO_STORAGE = 'thumbor.storages.file_storage'`

UPLOAD_DELETE_ALLOWED

Indicates whether image deletion should be allowed.

i.e.: `UPLOAD_DELETE_ALLOWED = False`

UPLOAD_PUT_ALLOWED

Indicates whether image overwrite should be allowed.

i.e.: `UPLOAD_PUT_ALLOWED = False`

UPLOAD_DEFAULT_FILENAME

Default filename for image uploaded.

i.e.: `UPLOAD_DEFAULT_FILENAME = 'image'`

GC_INTERVAL

Set manual garbage collection interval in seconds. Defaults to None (no manual garbage collection). Try this if your Thumbor is running out of memory. May cause an increase in CPU load.

i.e.: `GC_INTERVAL=60`

Example of Configuration File

```
################################### Logging ###################################

## Log Format to be used by thumbor when writing log messages.
## Defaults to: %(asctime)s %(name)s:%(levelname)s %(message)s
#THUMBOR_LOG_FORMAT = '%(asctime)s %(name)s:%(levelname)s %(message)s'

## Date Format to be used by thumbor when writing log messages.
## Defaults to: %Y-%m-%d %H:%M:%S
#THUMBOR_LOG_DATE_FORMAT = '%Y-%m-%d %H:%M:%S'

################################################################################


################################### Imaging ###################################

## Max width in pixels for images read or generated by thumbor
## Defaults to: 0
#MAX_WIDTH = 0

## Max height in pixels for images read or generated by thumbor
## Defaults to: 0
#MAX_HEIGHT = 0

## Min width in pixels for images read or generated by thumbor
## Defaults to: 1
#MIN_WIDTH = 1

## Min width in pixels for images read or generated by thumbor
## Defaults to: 1
#MIN_HEIGHT = 1

## Allowed domains for the http loader to download. These are regular
## expressions.
## Defaults to: []
#ALLOWED_SOURCES = #    [
#    ]


## Quality index used for generated JPEG images
## Defaults to: 80
#QUALITY = 80

## Max AGE sent as a header for the image served by thumbor in seconds
## Set to False to disable setting of Expires and Cache-Control headers
## Defaults to: 86400
#MAX_AGE = 86400

## Indicates the Max AGE header in seconds for temporary images (images that
## haven't been detected yet)
## Defaults to: 0
#MAX_AGE_TEMP_IMAGE = 0

## Indicates whether thumbor should rotate images that have an Orientation EXIF
## header
## Defaults to: False
#RESPECT_ORIENTATION = False

## Indicates whether thumbor should enable the EXPERIMENTAL support for animated
## gifs.
## Defaults to: True
#ALLOW_ANIMATED_GIFS = True

################################################################################


################################ Extensibility #################################

## The loader thumbor should use to load the original image. This must be the
## full name of a python module (python must be able to import it)
## Defaults to: thumbor.loaders.http_loader
#LOADER = 'thumbor.loaders.http_loader'

## The file storage thumbor should use to store original images. This must be the
## full name of a python module (python must be able to import it)
## Defaults to: thumbor.storages.file_storage
#STORAGE = 'thumbor.storages.file_storage'

## The result storage thumbor should use to store generated images. This must be
## the full name of a python module (python must be able to import it)
## Defaults to: None
#RESULT_STORAGE = None

## The imaging engine thumbor should use to perform image operations. This must
## be the full name of a python module (python must be able to import it)
## Defaults to: thumbor.engines.pil
#ENGINE = 'thumbor.engines.pil'

################################################################################


################################## Security ##################################

## The security key thumbor uses to sign image URLs
## Defaults to: MY_SECURE_KEY
#SECURITY_KEY = 'MY_SECURE_KEY'

## Indicates if the /unsafe URL should be available
## Defaults to: True
#ALLOW_UNSAFE_URL = True

## Indicates if encrypted (old style) URLs should be allowed
## Defaults to: True
#ALLOW_OLD_URLS = True

################################################################################


################################# File Loader ##################################

## The root path where the File Loader will try to find images
## Defaults to: /tmp
#FILE_LOADER_ROOT_PATH = '/tmp'

################################################################################


################################# File Storage #################################

## Expiration in seconds for the images in the File Storage. Defaults to one
## month
## Defaults to: 2592000
#STORAGE_EXPIRATION_SECONDS = 2592000

## Indicates whether thumbor should store the signing key for each image in the
## file storage. This allows the key to be changed and old images to still be
## properly found
## Defaults to: False
#STORES_CRYPTO_KEY_FOR_EACH_IMAGE = False

## The root path where the File Storage will try to find images
## Defaults to: /var/folders/th/z6vmj34j1gngpvwl5fg5t9440000gp/T/thumbor/storage
#FILE_STORAGE_ROOT_PATH = '/var/folders/th/z6vmj34j1gngpvwl5fg5t9440000gp/T/thumbor/
↪storage'

################################################################################
```

```
################################### Upload ###################################

## Max size in Kb for images uploaded to thumbor
## Aliases: MAX_SIZE
## Defaults to: 0
#UPLOAD_MAX_SIZE = 0

## Indicates whether thumbor should enable File uploads
## Aliases: ENABLE_ORIGINAL_PHOTO_UPLOAD
## Defaults to: False
#UPLOAD_ENABLED = False

## The type of storage to store uploaded images with
## Aliases: ORIGINAL_PHOTO_STORAGE
## Defaults to: thumbor.storages.file_storage
#UPLOAD_PHOTO_STORAGE = 'thumbor.storages.file_storage'

## Indicates whether image deletion should be allowed
## Aliases: ALLOW_ORIGINAL_PHOTO_DELETION
## Defaults to: False
#UPLOAD_DELETE_ALLOWED = False

## Indicates whether image overwrite should be allowed
## Aliases: ALLOW_ORIGINAL_PHOTO_PUTTING
## Defaults to: False
#UPLOAD_PUT_ALLOWED = False

## Default filename for image uploaded
## Defaults to: image
#UPLOAD_DEFAULT_FILENAME = 'image'

################################################################################


############################## MongoDB Storage ###############################

## MongoDB storage server host
## Defaults to: localhost
#MONGO_STORAGE_SERVER_HOST = 'localhost'

## MongoDB storage server port
## Defaults to: 27017
#MONGO_STORAGE_SERVER_PORT = 27017

## MongoDB storage server database name
## Defaults to: thumbor
#MONGO_STORAGE_SERVER_DB = 'thumbor'

## MongoDB storage image collection
## Defaults to: images
#MONGO_STORAGE_SERVER_COLLECTION = 'images'

################################################################################


############################### Redis Storage ################################
```

```
## Redis storage server host
## Defaults to: localhost
#REDIS_STORAGE_SERVER_HOST = 'localhost'

## Redis storage server port
## Defaults to: 6379
#REDIS_STORAGE_SERVER_PORT = 6379

## Redis storage database index
## Defaults to: 0
#REDIS_STORAGE_SERVER_DB = 0

## Redis storage server password
## Defaults to: None
#REDIS_STORAGE_SERVER_PASSWORD = None

################################################################################


############################## Memcache Storage ##############################

## List of Memcache storage server hosts
## Defaults to: ['localhost:11211']
#MEMCACHE_STORAGE_SERVERS = #    [
#         'localhost:11211',
#     ]


################################################################################


############################### Mixed Storage ################################

## Mixed Storage file storage. This must be the full name of a python module
## (python must be able to import it)
## Defaults to: thumbor.storages.no_storage
#MIXED_STORAGE_FILE_STORAGE = 'thumbor.storages.no_storage'

## Mixed Storage signing key storage. This must be the full name of a python
## module (python must be able to import it)
## Defaults to: thumbor.storages.no_storage
#MIXED_STORAGE_CRYPTO_STORAGE = 'thumbor.storages.no_storage'

## Mixed Storage detector information storage. This must be the full name of a
## python module (python must be able to import it)
## Defaults to: thumbor.storages.no_storage
#MIXED_STORAGE_DETECTOR_STORAGE = 'thumbor.storages.no_storage'

################################################################################


#################################### Meta ####################################

## The callback function name that should be used by the META route for JSONP
## access
## Defaults to: None
#META_CALLBACK_NAME = None
```

```
################################################################################


################################## Detection ###################################

## List of detectors that thumbor should use to find faces and/or features. All
## of them must be full names of python modules (python must be able to import
## it)
## Defaults to: []
#DETECTORS = #    [
#    ]


## The cascade file that opencv will use to detect faces
## Defaults to: haarcascade_frontalface_alt.xml
#FACE_DETECTOR_CASCADE_FILE = 'haarcascade_frontalface_alt.xml'

################################################################################


################################### Filters ####################################

## List of filters that thumbor will allow to be used in generated images. All of
## them must be full names of python modules (python must be able to import
## it)
## Defaults to: []
#FILTERS = #    [
#    ]


################################################################################


################################ Result Storage ################################

## Expiration in seconds of generated images in the result storage
## Defaults to: 0
#RESULT_STORAGE_EXPIRATION_SECONDS = 0

## Path where the Result storage will store generated images
## Defaults to: /var/folders/th/z6vmj34j1gngpvwl5fg5t9440000gp/T/thumbor/result_
↪storage
#RESULT_STORAGE_FILE_STORAGE_ROOT_PATH = '/var/folders/th/
↪z6vmj34j1gngpvwl5fg5t9440000gp/T/thumbor/result_storage'

## Indicates whether unsafe requests should also be stored in the Result Storage
## Defaults to: False
#RESULT_STORAGE_STORES_UNSAFE = False

################################################################################


########################### Queued Redis Detector ##############################

## Server host for the queued redis detector
## Defaults to: localhost
#REDIS_QUEUE_SERVER_HOST = 'localhost'
```

```
## Server port for the queued redis detector
## Defaults to: 6379
#REDIS_QUEUE_SERVER_PORT = 6379

## Server database index for the queued redis detector
## Defaults to: 0
#REDIS_QUEUE_SERVER_DB = 0

## Server password for the queued redis detector
## Defaults to: None
#REDIS_QUEUE_SERVER_PASSWORD = None

################################################################################


############################# Queued SQS Detector ##############################

## AWS key id
## Defaults to: None
#SQS_QUEUE_KEY_ID = None

## AWS key secret
## Defaults to: None
#SQS_QUEUE_KEY_SECRET = None

## AWS SQS region
## Defaults to: us-east-1
#SQS_QUEUE_REGION = 'us-east-1'

################################################################################


#################################### Errors ####################################

## This configuration indicates whether thumbor should use a custom error
## handler.
## Defaults to: False
#USE_CUSTOM_ERROR_HANDLING = False

## Error reporting module. Needs to contain a class called ErrorHandler with a
## handle_error(context, handler, exception) method.
## Defaults to: thumbor.error_handlers.sentry
#ERROR_HANDLER_MODULE = 'thumbor.error_handlers.sentry'

################################################################################


############################### Errors - Sentry ################################

## Sentry thumbor project dsn. i.e.: http://5a63d58ae7b94f1dab3dee740b301d6a:73ee
## a45d3e8649239a973087e8f21f98@localhost:9000/2
## Defaults to:
#SENTRY_DSN_URL = ''

################################################################################
```

Custom Error Handling

For many companies it make a lot of sense to have a centralized solution for handling errors in production, like sentry or squash.

Thumbor must support this type of error handling in order to better integrate itself to it's users environments.

Enabling Custom Error Handling

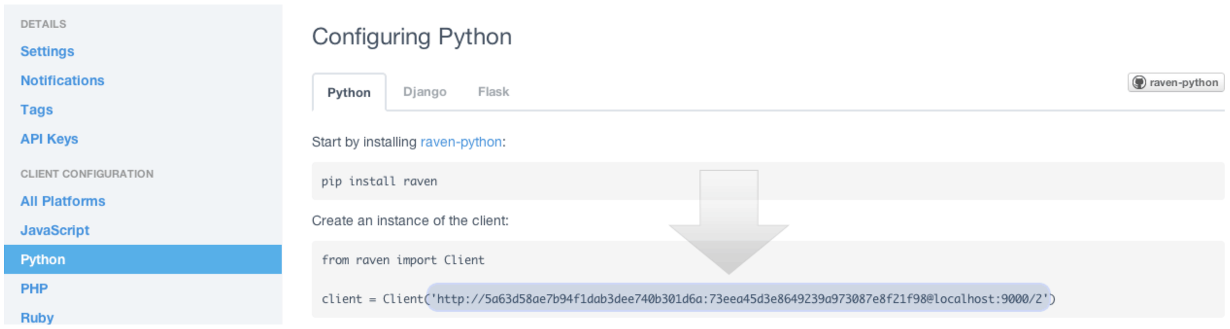
Enabling it is as simple as setting the configuration `USE_CUSTOM_ERROR_HANDLING` to `True`.

After that you need to set the custom error handler you want to use with the `ERROR_HANDLER_MODULE` configuration. Please note that this is the module full name, not the class full name.

Thumbor comes pre-packaged with sentry's custom error handler: `thumbor.error_handlers.sentry`. If you decide to use it, please read below on how to configure it.

Sentry - thumbor.error_handlers.sentry

If you choose to use sentry custom error handler, all you need to do is fill the `SENTRY_DSN_URL` configuration with sentry's DSN URL, which can be found in the admin page for your sentry project, like the one in the image below:



Configuring Python

Start by installing raven-python:

```
pip install raven
```

Create an instance of the client:

```
from raven import Client

client = Client('http://5a63d58ae7b94f1dab3dee740b301d6a:73eea45d3e8649239a973087e8f21f98@localhost:9000/2')
```

Writing my own Error Handler

Writing your own error handler is very simple. Just create a class called `ErrorHandler`, like the one below:

```python
class ErrorHandler(object):
    def __init__(self, config):
        # perform any initialization needed
        pass

    def handle_error(self, context, handler, exception):
        # do your thing here
        # context is thumbor's context for the current request
        # handler is tornado's request handler for the current request
        # exception is the error that occurred
```

When you have your handler done, just put it's full name in thumbor.conf and make sure thumbor can import it (it's somewhere in PYTHONPATH).

Hosting

Let's see how would you run thumbor in different environments.

Development Environment

For running it locally you just need to get a proper *Configuration* file. You can put it at /etc/thumbor.conf, ~/thumbor.conf (home folder) or specify it when starting thumbor.

To verify if you have thumbor, just type:

```
thumbor --version
```

It should return the version you've installed. Starting thumbor is as easy as:

```
thumbor
```

For more options check the *Configuration* page.

Production Environment

Other than having the proper *Configuration* file for your environment, we have some recommendations on how to run thumbor in production.

Our first recommendation is to run more than one instance of it. You can specify different ports using thumbor easily. This will make sure that your service stays responsive even if one of the processes die.

We also recommend having some form of load balance that distributes the load between the aforementioned processes. We are using NGINX to do it, but there are more sophisticated load balance softwares around. thumbor supports health checking under the `/healthcheck` URI if you need to use it.

Other than that, you run it using the thumbor console app specifying the arguments, like this:

```
thumbor --port=8888 --conf="~/mythumbor.conf"
```

We recommend using an application such as Supervisor (http://supervisord.org/index.html) to monitor your services. An example of a `supervisord.conf` file would be:

This configuration file makes sure that supervisor starts 4 processes of thumbor on the 8000, 8001, 8002 and 8003 ports, each with a different configuration file (thumbor8000.conf, thumbor8001.conf, thumbor8002.conf, thumbor8003.conf all under /etc folder). The other settings are optional, but if you need help with supervisor's settings it has extensive documentation online (http://supervisord.org/introduction.html).

Thumbor in the Cloud

UPDATE

Now there's a project to help with hosting in HEROKU called thumbor-heroku.

Creating your thumbor install in heroku

You can deploy and test Thumbor in the cloud. It's quite easy with Heroku :

- Create an account like described at http://devcenter.heroku.com/articles/quickstart
- Install the heroku Toolbelt as described in the same page
- Log to Heroku in your shell
- Create a small git project for the configuration of your Thumbor instance.

The whole script to deploy and start an instance :

```
mkdir heroku
cd heroku/
echo "thumbor>=2.7.0" >> requirements.txt # let heroku deploy and compile␣
↪prerequisite package via PIP
echo "web: thumbor -p $PORT" >> Procfile # listening port is automatically affected␣
↪at deployment (we use here the default config)
git init
git add .
git commit -m "init"
heroku create --stack cedar
git push heroku master
```

Basically, adding thumbor in requirements.txt will install everything you need on Heroku, and you just need to run thumbor -p $PORT to run thumbor on Heroku. In order to run process on Heroku, you need to write down the command in Procfile. Procfile looks like following (make sure there are no "" inside both files):

```
$ cat Procfile
web: thumbor -p $PORT
```

Your heroku folder (or whatever you named, I named it thumbor) should look like following (only contains two files):

```
~/thumbor(master)$ ls
Procfile        requirements.txt
```

- Start the instance (Remember: 1 heroku web instance is free of charges, so don't try with more yet):

  heroku scale web=1

- Verify your new instance is up (in the case of our sample project is stormy-stone-5336.herokuapp.com):

  heroku ps

- Now if you point your browser to the server name, you'll get a 404 HTTP Error. Just try with an URL that thumbor understands. To open your web browser pointing to the new server:

  heroku open

- Then try something like:

http://stormy-stone-5336.herokuapp.com/unsafe/300x200/http://s.glbimg.com/jo/g1/f/original/2012/03/16/supersonic-skydiver_fran.jpg

(notice there is no listening port specified)

If you need to scale thumbor server, read more about it in Heroku's documentation.

The sample implementation for the above links can be found at https://github.com/heynemann/thumbor-heroku and is open-source and MIT Licensed.

Another Thumbor/Heroku configuration

This blog post and the attached repositories (Jetpack and thumbor-heroku) explain a more advanced Heroku deployment, that support the `smart` URL feature.

Thumbor on OpenShift

There's a project showing how to deploy a working version on OpenShift https://github.com/rafaelcaricio/thumbor-openshift-example

Thumbor behind CloudFront

The awesome people at yipit are using thumbor behind the CloudFront CDN at Amazon.

The detailed information on how to do it can be seen at this blog post.

Logging

thumbor uses the built-in Python logging mechanisms. In order to configure log-level check the *Running thumbor server* page.

Configuring log format

Configuring the log format is as easy as including these keys in your `thumbor.conf` file:

THUMBOR_LOG_FORMAT

Log Format to be used by thumbor when writing log messages.

Defaults to: %(asctime)s %(name)s:%(levelname)s %(message)s

THUMBOR_LOG_DATE_FORMAT

Date Format to be used by thumbor when writing log messages.

Defaults to: %Y-%m-%d %H:%M:%S

Running thumbor server

Running thumbor server is as easy as typing "thumbor" (considering you went through the proper *Installing* procedures).

The Server application takes some parameters that will help you tailor the thumbor Server to your needs. If you want to find out what the thumbor Server arguments are, just type:

```
thumbor --help
```

-i or –ip

The address that Tornado will listen for incoming request. It defaults to *0.0.0.0* (listening on localhost and current IP).

-p or –port

The port that Tornado will listen for incoming request. It defaults to *8888*.

-c or –conf

The full path for the configuration file for this server.

-k or –keyfile

The full path for the file containing the security key to be used for this server.

-l or –log-level

The log level to be used. Possible values are: *debug*, *info*, *warning*, *error*, *critical* or *notset*. More on that at http://docs.python.org/library/logging.html. It defaults to *warning*.

-a or –app

Allows the user to specify the application class to be used. This is a very advanced usage of thumbor. This argument is specified like: "namespace1.namespace2.class_name" as in "myproj.thumbor_support.MyProjThumborApp".

Signing thumbor urls

To help users create signed URLs (mostly for debugging purposes, since we recommend using the *Libraries*), thumbor comes with an application called thumbor-url.

In order to use it, type `thumbor-url -h` and it will present all options available.

Scaling Thumbor

- Bernardo Heynemann wrote an article detailing this at Scaling Thumbor;
- Yipit has detailed how they scale thumbor at their engineering blog;
- Square also posted at their engineering blog about how they generate dynamic images with thumbor;
- 99 Designs also has some info on their architecture using thumbor and amazon at their engineering blog.
- Threadless wrote about how thumbor is enabling their mobile app to show proper thumbnails even when facing original images meant for printing in their engineering blog.

Security

thumbor's team is very concerned about security and vulnerabilities of the service. Even though the team strives to cover most scenarios, if you find any flaws or vulnerabilities, please contact the team or create an issue.

URL Tampering

Consider the following URL for an image: `http://some.server.com/unsafe/300x300/smart/path/to/image.jpg`.

Now let's say that some malicious user wants to overload your service. He can easily ask for other sizes in loops or worse, like:

```
http://some.server.com/unsafe/300x301/smart/path/to/image.jpg
http://some.server.com/unsafe/300x302/smart/path/to/image.jpg
http://some.server.com/unsafe/300x303/smart/path/to/image.jpg
...
http://some.server.com/unsafe/300x9999/smart/path/to/image.jpg
...
http://some.server.com/unsafe/9999x9999/smart/path/to/image.jpg
```

And that's not even counting varying the available options.

Other than that, the user can ask for images that do not exist, thus forcing us to perform useless http GET operations or filesystem operations.

We classified both scenarios above as **URL Tampering**.

Stopping Tampering

In order to prevent users from tampering with the URL, thumbor provides a configuration called `SECURITY_KEY`. This is the key used to generate a hash-based message authentication code.

The process is very straightforward. The web server that has the page using thumbor's image generates an authentication code for the options and image url, using the **SECURITY_KEY**.

When end-users access the page and thus load the image, thumbor generates an authentication code for the same options and image url, using the same **SECURITY_KEY**. If both authentication codes match, thumbor processes it.

The secure endpoint looks like this: `/<authentication code with 28 characters>/300x200/smart/path/to/image.jpg`.

HMAC method

We intend to supply toolkits in many languages that automate the signing process, but we might need help from the community in this direction.

thumbor uses standard HMAC with SHA1 signing.

Let's use as an example the url `http://some.server.com/unsafe/300x200/smart/path/to/image.jpg`.

In order to convert that to a "safe" url, we must sign the part `300x200/smart/path/to/image.jpg`:

1. Generate a `signature` of that part using HMAC-SHA1 with the **SECURITY_KEY**.
2. Encode the `signature` as base64. thumbor uses `urlsafe_b64encode` method of the native python's base64 module. This method replaces some characters in the base64 string so it becomes more url friendly.
3. Append the `encoded_signature` to the beginning of the URL, like: `/1234567890123456789012345678/300x200/smart/path/to/image.jpg`.

That last part gives you the new url: http://thumbor-server/1234567890123456789012345678/300x200/smart/path/to/image.jpg. Notice that the url includes the options part `300x200/smart`. That's required for thumbor to generate an authentication code to match the one that signs the image (`1234567890123456789012345678`).

The code included in this documentation is illustrational and should not be used for any purposes.

The description of the base64 method is: reference

```
base64.urlsafe_b64encode(s)
Encode string s using a URL-safe alphabet, which substitutes
- instead of + and _ instead of / in the standard Base64 alphabet.
The result can still contain =.
```

The old way

thumbor used to generate URL's differently using AES encryption/decryption. If you need more info on the old way of generating URLs, read the 3.0.0 release changes.

Loading Images over HTTPS

The default http_loader loads images by default over http. To change the default to https, use the https_loader instead. To enforce https, use the strict_https_loader. Check the *Image loader* page for more details.

Libraries

There are implementations of url generators in various languages, take a look at the [*Libraries*] page to find information about them.

More Information

- *Relase Notes*

Creating my own Storage

In order to create your own original photo storage, all you have to do is implement a class called `Storage` that inherits from `thumbor.storages.BaseStorage` and has three simple methods: `put`, `exists` and `remove`.

`put` is the method that actually stores the image somewhere. It could send the picture to a remote storage like Amazon's S3 or it could just save the picture to the local filesystem. This method should have a signature of `put(path, bytes)` and it should return the file path (for future reference).

`exists` should return if the file in the given path already exists. This method should have a signature of `exists(path)` and it should return a boolean stating if the file exists.

`remove` should just remove the file in the given path. This method *must* be idempotent, meaning that if the file has already been removed (or does not exist for that matter) it shouldn't do anything on subsequent calls. This method should have a signature of `remove(path)` and does not need to return anything.

After your class has been created (and hopefully tested, lol), you just need to modify the `ORIGINAL_PHOTO_STORAGE` configuration option in your thumbor.conf file to the module where you implemented your `Storage` class. Please note that thumbor must be able to import this module, so it should be somewhere in the PYTHONPATH you started thumbor with.

Extending

Plugins

With its pluggable architecture, thumbor provides extension points for a myriad of plug-in: storages, loaders, detectors, filters.

If your plug-in is not listed here, please create an issue with the details and we'll add it here.

Storages

thumbor_aws (by Thumbor Community)

Thumbor is a smart imaging service. It enables on-demand crop, resizing and flipping of images.

AWS is a cloud service, providing - among other things - storage capabilities.

This module provides support for AWS S3 interconnection, as a loader, a storage and/or a result storage.

- *URL:* https://github.com/thumbor-community/aws
- *Installing:* `pip install tc_aws`

To get exhaustive details about configuration options & setting it up, go to the documentation of the plugin.

thumbor_hbase (by Damien Hardy)

Thumbor is a smart imaging service. It enables on-demand crop, resizing and flipping of images.

Hbase is a column oriented database from the hadoop ecosystem.

This module provide support for Hadoop Hbase as large auto replicant key/value backend storage for images in Thumbor.

- *URL:* https://github.com/dhardy92/thumbor_hbase
- *Installing:* `pip install thumbor_hbase`

Using it is simple, just change your configuration in thumbor.conf:

```
HBASE_STORAGE_SERVER_HOST = "localhost"
HBASE_STORAGE_SERVER_PORT = 9000
HBASE_STORAGE_TABLE = "storage-table"
HBASE_STORAGE_FAMILY = "storage-family"
```

If you want to use thumbor_hbase for loading original images, change your thumbor.conf to read:

```
LOADER = "thumbor_hbase.loader"
```

If you want to use thumbor_hbase for storage of original images, change your thumbor.conf to read:

```
STORAGE = "thumbor_hbase.storage"
```

thumbor_mongodb (by Damien Hardy)

Thumbor is a smart imaging service. It enables on-demand crop, resizing and flipping of images.

MongoDB is a document oriented NoSQL database.

This plugin for Thumbor is a loader that can reach images from a mongodb collection based on its Object(_id).

- *URL:* https://github.com/dhardy92/thumbor_mongodb
- *Installing:* `pip install thumbor_mongodb`

Using it is simple, just change your configuration in thumbor.conf:

```
LOADER = 'thumbor_mongodb.loader'
MONGO_LOADER_CNX_STRING = 'mongodb://mongodbserver01,mongodbserver02:27017'
MONGO_LOADER_SERVER_DB = 'thumbor'
MONGO_LOADER_SERVER_COLLECTION = 'images'
MONGO_LOADER_DOC_FIELD = 'content'
```

thumbor_riak (by Damien Hardy)

Riak is a distributed document oriented database implementing the consistent hashing algorythm from the Dynanmo publication by Amazon.

This module provide support for Riak as a large auto replicant key/value backend storage for images in Thumbor.

- *URL:* https://github.com/dhardy92/thumbor_riak
- *Installing:* `pip install thumbor_riak` (require thumbor)

Using it is simple, just change your configuration in thumbor.conf:

```
# Use riak for storage.
STORAGE = 'thumbor_riak.storage'

# Put the url for your riak install here
RIAK_STORAGE_BASEURL = "http://my-riak-install-base-url"
```

thumbor_rackspace (by David Mann)

This plugin allows users to store objects in the Rackspace cloud for result storage.

- *URL:* https://github.com/CodingNinja/thumbor_rackspace
- *Installing:* `pip install thumbor_rackspace`

Using it is simple, just change your configuration in thumbor.conf:

```
# Use rackspace for result storage.
# For more info on result storage: https://github.com/thumbor/thumbor/wiki/Result-
↪storage
RESULT_STORAGE = 'thumbor_rackspace.result_storages.cloudfile_storage'

# Pyrax Rackspace configuration file location
RACKSPACE_PYRAX_CFG = /var/thumbor/.pyrax.cfg

# Result Storage options
RACKSPACE_RESULT_STORAGE_EXPIRES = True # Set TTL on cloudfile objects
```

```
RACKSPACE_RESULT_STORAGES_CONTAINER = "cloudfile-container-name"
RACKSPACE_RESULT_STORAGES_CONTAINER_ROOT = "/"
```

thumbor_ceph (by Laurent Barbe)

Ceph a distributed object store designed to provide excellent performance, reliability and scalability.

This module provide support for Ceph RADOS as backend storage for images.

- *URL:* https://github.com/ksperis/thumbor_ceph
- *Installing:* `apt-get install python-ceph && pip install thumbor_ceph`

Configuration in thumbor.conf:

```
################################# File Storage #################################
STORAGE = 'thumbor_ceph.storages.ceph_storage'
CEPH_STORAGE_POOL = 'thumbor'

#################################### Upload ####################################
UPLOAD_PHOTO_STORAGE = 'thumbor_ceph.storages.ceph_storage'

################################ Result Storage ################################
RESULT_STORAGE = 'thumbor_ceph.result_storages.ceph_storage'
CEPH_RESULT_STORAGE_POOL = 'thumbor'
```

For monitors and keys, the values used are those defined in the configuration file ceph.conf.

Metrics

thumbor_prometheus (by Simon Effenberg)

Prometheus a monitoring and alerting toolkit.

This module provide support for Prometheus as metrics collector.

- *URL:* https://github.com/thumbor-community/prometheus
- *Installing:* `pip install tc_prometheus`

Configuration in thumbor.conf:

```
################################# Extensibility #################################
METRICS = 'tc_prometheus.metrics.prometheus_metrics'

# optional with defaults
PROMETHEUS_SCRAPE_PORT = 8000 # Port the prometheus client should listen on
```

Extensions

thumborshortener (by Thumbor Community)

Thumbor is a smart imaging service. It enables on-demand crop, resizing and flipping of images.

This module provides URL shortening capabilities for Thumbor. It will create an API that can shorten a thumbor URL, and then routing capabilities to reroute the shortened URL to the correct image.

The shortened URL / real URL mapping is stored within redis.

- *URL:* https://github.com/thumbor-community/shortener
- *Installing:* `pip install tc_shortener`

To get exhaustive details about configuration options & setting it up, go to the documentation of the plugin.

Libraries

Even though the process of generating safe image URLs is explained in the *Security* page, we'll try to provide libraries in each programming language to ease this process.

Available Libraries

Python

- libthumbor - Python's extensions to thumbor. These are used to generate safe urls among others.
- django-thumbor - A django app with templatetags for resizing images with thumbor (by ricobl).
- django-thumborstorage - A Django custom storage for Thumbor backend (by Stanislas Guerra).

Node.js

- ThumborJS - Javascript's extension to thumbor. These are used to generate safe urls, encrypted urls among others (by Rafael Carício).
- ThumborUrlBuilder - Thumbor client for Node JS (by David Caramelo).
- thumbor - Thumbor client for Node JS (by PolicyMic).

Ruby

- ruby-thumbor - Ruby's gem to interact with thumbor server.
- thumbor_rails - Ruby's gem to make easier to generate urls in Rails projects.

Java

- Pollexor - Java client for the Thumbor image service which allows you to build URIs in an expressive fashion using a fluent API.
- thumbor-enterprise-edition - Java library to enable generating encrypted URLs. This library is deprecated in favor of Pollexor.

PHP

- Phumbor - A minimal PHP client for generating Thumbor URLs.
- Phumbor for Laravel - A Laravel package providing a facade for Phumbor.
- Phumbor for Symfony2 - A Symfony2 Bundle providing a facade for Phumbor.

Objective-C

- OCThumbor - Objective-C for the Thumbor image service which allows you to build URIs in an expressive fashion using a fluent API.

.NET

- DotNetThumbor - DotNet client for the Thumbor image service. Provides an expressive fluent API.

Implementing a library

If you want to provide a library to enable easy usage of thumbor in your favorite programming language, please send an e-mail to **thumbor@googlegroups.com** and we'll add it here.

Below are all the scenarios we think are worth testing automatically so you can guarantee compatibility with thumbor. Please note that this is not meant to be a replacement for TDD or for any other testing methodology you might want to use. These are just helper scenarios that we thought would help any library developers.

Library Tests - Generating HMAC of the URLs

We sincerely advise you to have thumbor installed in your machine, so you can implement a method in your tests that has thumbor generate a signature for your URL so you can compare with your own signature. This way you can make sure your url formatting and signing are working properly.

Here's how it was implemented in Ruby:

```ruby
def sign_in_thumbor(key, str)
    #bash command to call thumbor's decrypt method
    command = "python -c 'from thumbor.crypto import Signer; signer = Signer(\"" << key << "\"); print signer.signature(\"" << str << "\")'"

    #execute it in the shell using ruby's popen mechanism
    result = Array.new
    IO.popen(command) { |f| result.push(f.gets) }

    result.join('')
end
```

You should be able to implement this easily in any modern programming language. It makes for very reliable tests.

Library Tests - Scenarios

Remember that these are in pseudo-code (BDD-like) language, and not in any programming language specifically.

Encryption Testing

These scenarios assume that you separate the logic of composing the url to be signed into a different "module", that is to be tested with the URL Testing Scenarios after these scenarios.

Scenario 1 - Signing of a known url results

```
Given
    A security key of 'my-security-key'
    And an image URL of "my.server.com/some/path/to/image.jpg"
    And a width of 300
    And a height of 200
When
    I ask my library for a signed url
Then
    I get '/8ammJH8D-7tXy6kU3lTvoXlhu4o=/300x200/my.server.com/some/path/to/image.jpg
↪' as url
```

Scenario 2 - Thumbor matching of signature with my library signature

```
Given
    A security key of 'my-security-key'
    And an image URL of "my.server.com/some/path/to/image.jpg"
    And a width of 300
    And a height of 200
When
    I ask my library for an encrypted URL
Then
    I get the proper url (/8ammJH8D-7tXy6kU3lTvoXlhu4o=/300x200/my.server.com/some/
↪path/to/image.jpg)
```

Scenario 3 - Thumbor matching of signature with my library signature with meta

```
Given
    A security key of 'my-security-key'
    And an image URL of "my.server.com/some/path/to/image.jpg"
    And the meta flag
When
    I ask my library for an encrypted URL
Then
    I get the proper url (/Ps3ORJDqxlSQ8y00T29GdNAh2CY=/meta/my.server.com/some/path/
↪to/image.jpg)
```

Scenario 4 - Thumbor matching of signature with my library signature with smart

```
Given
    A security key of 'my-security-key'
    And an image URL of "my.server.com/some/path/to/image.jpg"
    And the smart flag
When
    I ask my library for an encrypted URL
Then
    I get the proper url (/-2NHpejRK2CyPAm61FigfQgJBxw=/smart/my.server.com/some/path/
↪to/image.jpg)
```

Scenario 5 - Thumbor matching of signature with my library signature with fit-in

```
Given
    A security key of 'my-security-key'
    And an image URL of "my.server.com/some/path/to/image.jpg"
    And the fit-in flag
When
    I ask my library for an encrypted URL
Then
    I get the proper url (/uvLnA6TJlF-Cc-L8z9pEtfasO3s=/fit-in/my.server.com/some/
↪path/to/image.jpg)
```

Scenario 6 - Thumbor matching of signature with my library signature with filters

```
Given
    A security key of 'my-security-key'
    And an image URL of "my.server.com/some/path/to/image.jpg"
    And a 'quality(20)' filter
    And a 'brightness(10)' filter
When
    I ask my library for an encrypted URL
Then
    I get the proper url (/ZZtPCw-BLYN1g42Kh8xTcRs0Qls=/
↪filters:brightness(10):contrast(20)/my.server.com/some/path/to/image.jpg)
```

You should test the same kind of tests for horizontal and vertical flip, horizontal and vertical alignment and manual cropping.

More Information

- *Security*

More

The team

These are the people that created, that work or worked in thumbor across the releases:

Founders / Committers

- @heynemann - Founder and active committer
- @cezarsa - Active committer
- @fabiomcosta - Founder and active committer

Contributors

Contributors can be found in the Contributors page.

Whos using it

http://www.globo.com - globo.com uses thumbor to generate dynamic images through all products across the portal. Around 15 billion images served per month.

http://www.properati.com.ar/ - properati is also using thumbor to generate several different sizes of their properties photos, using smart cropping to get the best possible thumbnails.

> Thumbor made our lives better.
>
> At Properati.com.ar we care a lot about user experience.
>
> When our design team came up with a beautiful design that included thumbnails of 5 different sizes and specific cropping specs, instead of enhancing our home-made, simple thumbnail-generator process we moved to Thumbor and now, we cannot live without it.
>
> Thanks a lot!

http://yipit.com/ - yipit now uses thumbor behind the CloudFront CDN at Amazon. Their detailed experience with setting up thumbor can be seen at this blog post.

> Thumbor allows Yipit to iterate quickly on new designs without having to worry about introducing new image sizes.
>
> On demand image generation was just too slow when integrated into our application servers, but Thumbor makes it possible.
>
> No more going through old images and creating new thumbnails before we can roll out a new design!
>
> Our initial Thumbor installation took less than an hour to set up, and we haven't had to spend much time thinking about it since then.
>
> Zach Smith - CTO

http://oony.com is using thumbor to serve thumbnail images behing Amazon's Cloudfront CDN.

> We've previously adapted the size of the thumbnails to what was required by our design team, forcing us to have many different versions of the images we have on our site.
>
> With Thumbor we don't have to worry about this anymore, and we can quickly iterate and make changes to our layouts serving the optimal image format each time.
>
> Thumbor is awesome!

The Viadeo Group owns and operates professional social networks around the world with a total membership base of over 55 million professionals. Professionals use the networks to enhance their career prospects, discover business opportunities and build relationships with new contacts as well as to create effective online identities.

> With headquarters based in Paris, the Group currently has over 450 staff with offices and teams located in 10 countries.
>
> Viadeo is using Thumbor more and more. We used to have some home-made Java code to deliver images from http://www.viadeo.com. This code is still alive for some parts of our site.
>
> Since the end of summer 2013, Thumbor is a reality at Viadeo. First via IOS application for member's profile photos, then our news platform uses it for new parts of the site, taking more and more place and also some Android applications.
>
> Thumbor helps us in migrating and decoupling applications from our storage backend. We were able to move from NFS (centralized and very sensitive to high loads) to a distributed storage like HBase, using a hbase storage plugin. Using the same technique of lazy loading (via Storage cache in thumbor) we can imagine changing our image's storage at our convenience should apache HBase start to show deficiencies. This is really comfortable for Ops.

TypeTees is an easy-to-use iPhone app that lets you speak your mind by putting your super witty slogan into an original tee and order it immediately.

> We use Thumbor to generate mobile thumbnails directly from the same large images that are sent to the t-shirt garment printer. It requires dealing with masks, feature trimming, transparent images, and replacing backgrounds to give users an easy-to-see preview of the t-shirt.
>
> Thumbor made this possible and simple without having to write an image processor from scrap.
>
> TypeTees was developed by www.prolificinteractive.com and you can learn more about how thumbor helped them at their engineering blog post.

Just Watch

> At JustWatch, we're big fans of Thumbor as well.
>
> We're serving it behind a CloudFront custom origin like many others, and features like WebP and smart cropping saved us huge amounts of time and bandwidth.

Ridelink

> RideLink uses Thumbor to provide the most appropriate, and optimized, image for the platform our customer is using.
>
> Setting it up and enhancing it was an easy task thanks to the good documentation and the available plugins.
>
> Thanks to the team behind Thumbor for making our lifes easier!
>
> Cheers! Erico Andrei - CTO

How to add my site or product here

If you are using thumbor and your site or product is not listed here, please create an issue and we'll include your logo and a short description on how you are using it here.

Hacking on Thumbor

So you want to contribute with thumbor? Welcome onboard!

There are a few things you'll need in order to properly start hacking on it.

First step is to fork it and create your own clone of thumbor.

Dependencies

We seriously advise you to use virtualenv since it will keep your environment clean of thumbor's dependencies and you can choose when to "turn them on".

You'll also need python >= 2.6 and < 3.0.

The following packages are required:

- Tornado >= 2.1.1
- pyCrypto >= 2.4.1
- pycurl >= 7.19.0
- Pillow >= 1.7.5
- redis >= 2.4.11
- pymongo >= 2.1.1
- argparse

You'll also need a recent version of OpenCV installed. When installing OpenCV, it will create a python binding. Make sure this binding is visible to your current virtualenv (if you are using it).

Other than that, you'll also need a mongo database running, as well as a redis database running. Both are trivial to setup at modern linux or mac os systems.

Running the Tests

Running the tests is as easy as:

```
make test
```

You should see the results of running your tests after an instant.

If you are experiencing "Too many open files" errors while running the tests, try increasing the number of open files per process, by running this command:

```
ulimit -S -n 2048
```

Read http://superuser.com/questions/433746/is-there-a-fix-for-the-too-many-open-files-in-system-error-on-os-x-10-7-1 for more info on this.

Pull Requests

After hacking and testing your contribution, it is time to make a pull request. Make sure that your code is already integrated with the master branch of thumbor before asking for a pull request.

To add thumbor as a valid remote for your repository:

```
git remote add thumbor git://github.com/thumbor/thumbor.git
```

To merge thumbor's master with your fork:

```
git pull thumbor master
```

If there was anything to merge, just run your tests again. If they pass, send a pull request.

Licensing

Thumbor is licensed under the MIT License:

Relase Notes

** THIS PAGE IS HERE ONLY FOR HISTORIC PURPOSES, since we are now using github releases page. **

Stable Release

4.1.0 - http://pypi.python.org/pypi/thumbor/4.1.0 - 02-Apr-2014

thumbor Releases

4.1.0 - http://pypi.python.org/pypi/thumbor/4.1.0 - 02-Apr-2014 - diff

- New filter: Extract Focal Points
- Infrastructure for filters on different phases during the image processing lifecycle. Right now PHASE_POST_TRANSFORM and PHASE_PRE_LOAD are supported. All existing filters default to PHASE_POST_TRANSFORM

4.0.4 - http://pypi.python.org/pypi/thumbor/4.0.4 - 28-Mar-2014 - diff

- Fixed issue with blur filter when used with more than 150 of radius (by @heynemann);
- Fixed issue with format filter when used in conjunction with auto_webp (by @cezarsa).

4.0.3 - http://pypi.python.org/pypi/thumbor/4.0.3 - 28-Mar-2014 - diff

- Fix in all engines to return both image data and image mode together, instead of one or the other. If you implement your own engine, you need to create a new method called `image_data_as_rgb` that returns the image mode and the image bytes;
- Thumbor Application class now has a `get_handlers` method that can be overwritten to specify new handlers.

4.0.2 - http://pypi.python.org/pypi/thumbor/4.0.2 - 18-Mar-2014 - diff

- Fixed issue with WebP request path by Frank Du;
- Fixed issue with Upstart Script Log Level by Matt Robenolt;
- Fixed issue with folder not existing before storing security details by Cícero Verneck Corrêa;
- Fixed #272 - Thumbor works properly with newer tornado.

4.0.1 - http://pypi.python.org/pypi/thumbor/4.0.1 - 18-Mar-2014 - diff

- Fixed issue #289 - Now URLs with '~' should work properly.

4.0.0 - http://pypi.python.org/pypi/thumbor/4.0.0 - 12-Mar-2014 - diff

WARNING

This version contains breaking changes. Both GraphicsMagick and OpenCV engines were removed from the built-in imaging engines and can be found in the thumbor organization now. For more information on using each of them check the respective project documentation.

BREAKING Chances

- Removed thumbor.engines.opencv in favor of the new opencv-engine project. That's where we'll maintain the OpenCV engine.
- Removed thumbor.engines.graphicsmagick in favor of the new graphicsmagick-engine project. That's where we'll maintain the GraphicsMagick engine.

Fixed Issues

- Partitioning the FileStorage Result Storage into more folders by Martin Sarsale;
- Json File Error Handler by Damien Hardy;
- Support binding socket to file descriptor instead of port by John MacKenzie;

- HEAD queries to thumbor's healthcheck returning 200 status code by Damien Hardy;
- Fixed bug when parsing urls with filters of an original with filters by Cezar Sá;
- Support different default quality for WebP images by Bernardo Heynemann;
- Keep transparency when saving transparent gif by Igor Sobreira;
- Don't save PNG files as CMYK by Igor Sobreira;
- Upstart now uses ip var defined on ubuntu default file by Paulo Sousa;
- Fixed images cropped with width 1px and height 0px by Bernardo Heynemann;
- Fixed #236 - IndexError: list index out of range by Bernardo Heynemann;
- Fixed #235 - ValueError: Not a valid numbers of quantization tables. Should be between 2 and 4 by Bernardo Heynemann;
- Fixed #228 - Confusing error when using OpenCV by Bernardo Heynemann;
- New options to the fill filter by prolificphotis;
- Added FILL_MERGES Configuration to specify whether the fill filter should merge the background by prolificphotis;
- Resolved quality config None in graphicsmagick engine by Marcio Toshio Ide;
- Preserving EXIF info when storing original images by Cezar Sá;
- Resetting EXIF orientation after reorienting image by Cezar Sá;
- Compatibility work for the fill filter across engines by Cezar Sá;
- Pillow test_requirement match setup.py by Rob Olson;
- Fixed issues with graphicsmagick and gif images by Bernardo Heynemann;
- Convert to grayscale working in OpenCV Engine by Pablo Aguiar.

New features

- New convolution filter by Cezar Sá;
- New Gaussian Blur filter by Cezar Sá;

3.14.7 - http://pypi.python.org/pypi/thumbor/3.14.7 - 30-Oct-2013 - diff

- Bumping tornado version to allow last update.

3.14.6 - http://pypi.python.org/pypi/thumbor/3.14.6 - 07-Oct-2013 - diff

- Result storage disallows requesting files outside the root path.

3.14.5 - http://pypi.python.org/pypi/thumbor/3.14.5 - 25-Sep-2013 - diff

- Not doing vary header or converting to WebP when image is an animated gif or already a WebP.

3.14.4 - http://pypi.python.org/pypi/thumbor/3.14.4 - 24-Sep-2013 - diff

- Thumbor now includes a "Vary": "Accept" header to help cache servers to better understand that the image URL can vary by accept header.

3.14.1 - http://pypi.python.org/pypi/thumbor/3.14.1 - 02-Sep-2013 - diff

- A new filter has been introduced: max_bytes. This filter allows users to specify the maximum number of bytes for the image. Thumbor will vary the quality of the image for JPEG and WebP images (png and gif images do not get affected by this filter).

3.13.3 - http://pypi.python.org/pypi/thumbor/3.13.3 - 31-Aug-2013 - diff

- Fixed #193. File storage now uses atomic storage of files, thus avoiding corruption of stored images.

3.13.2 - http://pypi.python.org/pypi/thumbor/3.13.2 - 31-Aug-2013 - diff

- Merged #202. Proxy support added to default HTTP Loader.

3.13.1 - http://pypi.python.org/pypi/thumbor/3.13.1 - 31-Aug-2013 - diff

- Merged #197. Healthcheck now replied to HEAD requests.

3.13.0 - http://pypi.python.org/pypi/thumbor/3.13.0 - 28-Aug-2013 - diff

- Fixes #204. Thumbor now allows users to specify that WebP should be automatically used whenever the request has the proper Accept header (image/webp).

3.12.2 - http://pypi.python.org/pypi/thumbor/3.12.2 - 12-Aug-2013 - diff

- Added some extra logging to the finish request stage of the image handling.

3.12.1 - http://pypi.python.org/pypi/thumbor/3.12.1 - 18-Jul-2013 - diff

- Fixed leak of Redis connections when using queued detectors.

3.12.0 - http://pypi.python.org/pypi/thumbor/3.12.0 - 05-Jul-2013 - diff

- Fixed an issue with animated gifs (sigh);
- Add detection support for WEBP format. Merge pull request #194 from dhardy92:feature_Add_WEBP_Detection;
- Support for the new release of Pillow (2.1.0) and works with Pillow master branch for now.

3.11.1 - http://pypi.python.org/pypi/thumbor/3.11.1 - 05-Jul-2013 - diff

- Finished webp support;
- Fixed a bug with webp support that would pass 'None' as format if no format specified;
- Added a configuration `PRESERVE_EXIF_INFO` that when set to True will keep the exif metadata in images intact (including webp resulting images).

3.11.0 - http://pypi.python.org/pypi/thumbor/3.11.0 - 02-Jul-2013 - diff

- Added 'format' filter. Now users can specify the output format using filters:format(webp) or filters:format(jpeg) and as follows. More information in the Filters page.
- Partial webp support. Now webp images can be read as the source image and be used as the output image. Partial here means that the version we are using of pillow does not yet support ICC Profiles in WebP images. Only Chrome Canary does support ICC profiles right now, so this is not a real issue.
- Improved openCV engine image resampling.
- Proper integration with Pillow version 2.0.0.
- Fixed HMAC signing if the key has unicode characters.

3.10.0 - http://pypi.python.org/pypi/thumbor/3.10.0 - 14-May-2013 - diff

- Fixes #184. Thumbor now reports expected errors as warning, instead of errors. This should allow users to use a logger level of ERROR to reduce the amount of I/O thumbor does for logging.
- Fixes #183.
- Fixes #182. There's two new configuration keys: `HTTP_LOADER_DEFAULT_USER_AGENT` and `HTTP_LOADER_FORWARD_USER_AGENT`. These are meant to allow scenarios where the remote image server won't allow thumbor's user agent.
- Fixes #180. Thumbor now features a grayscale filter. More information can be found in the Filters page.
- Code reformatting to conform to PEP-8.

3.9.4 - http://pypi.python.org/pypi/thumbor/3.9.4 - 17-Apr-2013 - diff

- Upgraded Pillow dependency to 2.0.0;
- Normalized the Max Age header for images with smart detection errors around all detectors. Also included the `IGNORE_SMART_ERRORS` setting that enables users to keep responding the image without smart cropping when smart detection throws exceptions. This setting is `False` by default and needs to be enabled explicitly (reverse compatibility);
- Fixed an issue with sentry error handler;
- **POSSIBLE BREAKING CHANGE**: We changed the way the http handler requests images. It now passes safer connection timeout, request timeout and follow redirects values to `libcurl`. You can change those values in your configuration file using the `HTTP_LOADER_CONNECT_TIMEOUT`, `HTTP_LOADER_REQUEST_TIMEOUT`, `HTTP_LOADER_FOLLOW_REDIRECTS` and `HTTP_LOADER_MAX_REDIRECTS` settings (more on those in the Configuration page). This change might break you if you have connect times greater than 5 seconds. This setting was previously configured to 20 seconds.

3.9.2 - http://pypi.python.org/pypi/thumbor/3.9.2 - 09-Apr-2013 - diff

- Logging format can now be configured using `THUMBOR_LOG_FORMAT` and `THUMBOR_LOG_DATE_FORMAT` configuration variables. These are just passed through to python's `format` and `datefmt` arguments of the `logging.basicConfig` method.

3.9.1 - http://pypi.python.org/pypi/thumbor/3.9.1 - 09-Apr-2013 - diff

- Makes error handling a little safer.

3.9.0 - http://pypi.python.org/pypi/thumbor/3.9.0 - 28-Mar-2013 - diff

- Fixes #165. Setting the `ALLOW_ANIMATED_GIFS` configuration to `False` will remove the experimental support for animated gifs.

3.8.1 - http://pypi.python.org/pypi/thumbor/3.8.1 - 27-Mar-2013 - diff

- Fixes #175. Thumbor now support custom error handling. This can be very useful for users that have a centralized error application (like sentry).
- Sentry's custom error handler comes built-in with thumbor.
- Optimized fill filter, which is now implemented in C (by fabiomcosta).

3.7.1 - http://pypi.python.org/pypi/thumbor/3.7.1 - 06-Feb-2013 - diff

- Fix bug with quoting valid characters in URL (by cdemonchy);
- Fix in debian packaging for Debian Squeeze (by dhardy92);
- Fix in the mongo storage (by phpconnect);
- Auto option for the fill filter (by fabiomcosta).

3.7.0 - http://pypi.python.org/pypi/thumbor/3.7.0 - 24-Jan-2013 - diff

- Multi-Instance deb support. Merge pull request #146 from nhuray/master.

3.6.11 - http://pypi.python.org/pypi/thumbor/3.6.11 - 23-Jan-2013 - diff

- Implementing methods that were missing in the json engine;
- Merge pull request #143 from nhuray/master;
- Disable REST Upload by default;
- Merge pull request #142 from morpheu/master;
- Other detector options in thumbor.conf.

3.6.10 - http://pypi.python.org/pypi/thumbor/3.6.10 - 14-Dec-2012 - diff

- Fixes #138. Filters are not required for using thumbor.

3.6.9 - http://pypi.python.org/pypi/thumbor/3.6.9 - 12-Dec-2012 - diff

- Improved error handling on http loader.

3.6.8 - http://pypi.python.org/pypi/thumbor/3.6.8 - 12-Dec-2012 - diff

- Fixes #139. Libmagic is not required anymore.
- Improved image type detection.

3.6.7 - http://pypi.python.org/pypi/thumbor/3.6.7 - 24-Oct-2012 - diff

- Pull request #133 from gcirne.
- Fixes #132. Thumbor has a rest API for uploading images from this version onwards. Documentation to follow.

3.6.6 - http://pypi.python.org/pypi/thumbor/3.6.6 - 24-Oct-2012 - diff

- Fixed some issues with thumbor-url.

3.6.4 - http://pypi.python.org/pypi/thumbor/3.6.4 - 24-Oct-2012 - diff

- Fix glasses detector - Pull request #124.
- Pull request #128 from wichert.
- Update encrypted string to allow trim parameter;
- Allow specifying trim option in URL composure and thumbor-url.

3.6.3 - http://pypi.python.org/pypi/thumbor/3.6.3 - 26-Sep-2012 - diff

- Fixes #127.

3.6.2 - http://pypi.python.org/pypi/thumbor/3.6.2 - 19-Sep-2012 - diff

- Fixes #126.

3.6.1 - http://pypi.python.org/pypi/thumbor/3.6.1 - 19-Sep-2012 - diff

- Fixes #125 properly. Both libthumbor and ruby-thumbor verified now (Big Kudos to @robolson).

3.6.0 - http://pypi.python.org/pypi/thumbor/3.6.0 - 18-Sep-2012 - diff

- Fixed compilation under clang (Mac OS X Lion);
- Included trim option to remove surrounding space in images more info;
- Fixes #125.
- Pull request #124.

3.5.2 - http://pypi.python.org/pypi/thumbor/3.5.2 - 14-Aug-2012 - diff

- Fixed support to custom apps;
- Fixed issue with graphicsmagick manual crop method;
- Added a custom-header to thumbor that specifies its name and version;
- Changed filestorage to store uploaded files using a MD5 based hash algorithm similar to what git does.

3.5.1 - http://pypi.python.org/pypi/thumbor/3.5.1 - 03-Aug-2012 - diff

- Added a new exception in the upload handler called `BadRequestError` as a way for storages to report to thumbor that some information that they required in the request was not provided. This way thumbor can return a `400 BAD REQUEST` response to the upload request.

3.5.0 - http://pypi.python.org/pypi/thumbor/3.5.0 - 03-Aug-2012 - diff

- Fixes #113 and #114, that were related.
- Allow storage classes to retrieve request information in the `resolve_original_path` method.

WARNING - This release introduces a BREAKING CHANGE if you have your own storage implemented. The method `resolve_original_photo_path` now has a new signature. It used to be `resolve_original_photo_path(filename)` and now is `resolve_original_photo_path(request, filename)`.

3.4.1 - http://pypi.python.org/pypi/thumbor/3.4.1 - 02-Aug-2012 - diff

- Fixes #115.

3.4.0 - http://pypi.python.org/pypi/thumbor/3.4.0 - 01-Aug-2012 - diff

- Fixes #107. 9-Patch filter to support android 9-patch format-like images.
- Fixes #103. Fixes handling special characters in the URLs.
- A couple configuration keys renamed. For some time the old names will be kept compatible.
- Introduction of https://github.com/globocom/derpconf, an abstraction for configuration files.

3.3.0 - http://pypi.python.org/pypi/thumbor/3.3.0 - 18-Jul-2012

- Fixes #82. There's a new command called 'thumbor-config' that will output thumbor's default configuration file.
- Fixes #94. There's a new configuration called 'RESPECT_ORIENTATION' that instructs thumbor to rotate images according to an EXIF orientation (if one can be found in the image headers).

3.2.0 - http://pypi.python.org/pypi/thumbor/3.2.0 - 18-Jul-2012

- Fixes #103. Tornado unquotes URL's passed to thumbor and that screws up some URLs.

3.1.1 - http://pypi.python.org/pypi/thumbor/3.1.1 - 17-Jul-2012

- Fixes #102. There was an additional issue with images with alpha channels (LA).

3.1.0 - http://pypi.python.org/pypi/thumbor/3.1.0 - 17-Jul-2012

- Fixed issue with gifsicle when optimizing GIF images.
- Fixes #102. This was an issue with OpenCV and palette images.
- Fixes with URL regexes.

3.0.2 - http://pypi.python.org/pypi/thumbor/3.0.2 - 9-Jul-2012

- Fixing size and manual crop for animated gifs.

3.0.1 - http://pypi.python.org/pypi/thumbor/3.0.1 - 2-Jul-2012

Some fixes: * Fixed issue with filters in old style URLs. * Supporting meta in the thumbor-url console. * Using storage crypto keys for hmac.

3.0.0 - http://pypi.python.org/pypi/thumbor/3.0.0 - 2-Jul-2012

This release features a major change in the way URLs are handled. It's still backwards compatible, but the old style URLs are deprecated and will go away in the next major. For more information read the 3.0.0 release changes.

- Fixes #98.

2.8.2 - http://pypi.python.org/pypi/thumbor/2.8.2 - 9-Jul-2012

- Fixing size and manual crop for animated gifs. (Backport from 3.0.2)

2.8.1 - http://pypi.python.org/pypi/thumbor/2.8.1 - 29-Jun-2012

- Fixes #97. Request parameters for the source image are now properly appended to the image URI.
- Fixes #96. Experimental support for animated gifs. Most filters are working. Only for PIL engine. Other engines to come.

2.7.8 - http://pypi.python.org/pypi/thumbor/2.7.8 - 21-Jun-2012

- Fixes to the fill and watermark filters.

2.7.7 - http://pypi.python.org/pypi/thumbor/2.7.7 - 01-Jun-2012

- New filter to strip ICC heders
- Issue with ORIG size and Max Height.
- Encoding issues for Unicode named images.

2.7.4 - http://pypi.python.org/pypi/thumbor/2.7.4 - 30-Mar-2012

- Support to "orig" style widths and heights.

2.7.3 - http://pypi.python.org/pypi/thumbor/2.7.3 - 23-Mar-2012

- Issue #90|https://github.com/thumbor/thumbor/issues/90 fixed. thumbor-url command now works properly.
- Key file and adaptive cropping support in thumbor-url.

2.7.1 - http://pypi.python.org/pypi/thumbor/2.7.1 - 19-Mar-2012

- Filter infrastructure refactored.

2.7.0 - http://pypi.python.org/pypi/thumbor/2.7.0 - 14-Mar-2012

- Improvements in the upload feature.
- Improvements in the C-Based filters.

2.6.12 - http://pypi.python.org/pypi/thumbor/2.6.12 - 05-Mar-2012

- New sharpen filter.

2.6.5 - http://pypi.python.org/pypi/thumbor/2.6.5 - 01-Mar-2012

- Fixed issue with fill filter.

2.6.4 - http://pypi.python.org/pypi/thumbor/2.6.4 - 23-Feb-2012

- Minor fixes in the red eye and equalize filters.

2.6.3 - http://pypi.python.org/pypi/thumbor/2.6.3 - 21-Feb-2012

- Minor fixes in the image uploading area.

2.6.2 - http://pypi.python.org/pypi/thumbor/2.6.2 - 20-Feb-2012

- Ticket #25 in experimental status.
- Ticket #59 done.

2.5.1 - http://pypi.python.org/pypi/thumbor/2.5.1 - 02-Feb-2012

- Better handling errors in queued detectors;
- Fallback to jpeg when we don't know the image type;
- Increased test coverage.

2.5.0 - http://pypi.python.org/pypi/thumbor/2.5.0 - 30-Jan-2012

- Refactored base detector not to depend on opencv anymore.

2.4.9 - http://pypi.python.org/pypi/thumbor/2.4.9 - 30-Jan-2012

- Atomic file move for ResultStorage.

2.4.7 - http://pypi.python.org/pypi/thumbor/2.4.7 - 27-Jan-2012

- Bug fixes.
- Password support for redis storage.

2.4.6 - http://pypi.python.org/pypi/thumbor/2.4.6 - 24-Jan-2012

- Bug fixes in Mongo and Redis Storages.

2.4.4 - http://pypi.python.org/pypi/thumbor/2.4.4 - 18-Jan-2012

- Minor fixes in file descriptor management.

2.4.3 - http://pypi.python.org/pypi/thumbor/2.4.3 - 18-Jan-2012

- New setting that allows users to specify if unsafe images should be in result storage.

2.4.2 - http://pypi.python.org/pypi/thumbor/2.4.2 - 17-Jan-2012

- Minor tweaks to result storage.

2.4.1 - http://pypi.python.org/pypi/thumbor/2.4.1 - 17-Jan-2012

- Internal minor refactoring.

2.4.0 - http://pypi.python.org/pypi/thumbor/2.4.0 - 17-Jan-2012

- Major refactoring of thumbor internals. Should not affect thumbor usage.

2.3.0 - http://pypi.python.org/pypi/thumbor/2.3.0

- Features a RemoteCompleteDetector to perform both detections in one round-trip to remotecv.

2.2.0 - http://pypi.python.org/pypi/thumbor/2.2.0

- Included support for remotecv.

2.1.0 - http://pypi.python.org/pypi/thumbor/2.1.0

- Updated tornado to release 2.1.1.

2.0.5 - http://pypi.python.org/pypi/thumbor/2.0.5

- Improved PIL graphics engine to support different ICC profiles. It now keeps the existing ICC profile if there is one. This improves drastically the image quality. Very recommended update.

2.0.3 - http://pypi.python.org/pypi/thumbor/2.0.3

- Fixes to native extensions used in filters.

2.0.2 - http://pypi.python.org/pypi/thumbor/2.0.2

- Fixed issue with specifying the jsonp callback.

2.0.1 - http://pypi.python.org/pypi/thumbor/2.0.1

- Debug mode.
- Filter Support.
- Brightness, Contrast, Noise, Quality, RGB, Round Corner and Watermark filters.
- ImageMagick engine removed.

- JSONP callback can now be passed as an argument.
- Minor fixes.

1.2.1 - http://pypi.python.org/pypi/thumbor/1.2.1

- Fixed minor issues with storing openCV results.

1.1.0 - http://pypi.python.org/pypi/thumbor/1.1.0

- Fixed bug with smart cropping manual cropped images.

1.0.0 - http://pypi.python.org/pypi/thumbor/1.0.0

- Fixed major bug with manual cropping.

0.9.6 - http://pypi.python.org/pypi/thumbor/0.9.6

- Crypto Handler refactored. Improved decrypting performance.

0.9.4 - http://pypi.python.org/pypi/thumbor/0.9.4

- Fixing the number of processes to one.

0.9.3 - http://pypi.python.org/pypi/thumbor/0.9.3

- Fixes issue with mysql storage.

0.9.1 - http://pypi.python.org/pypi/thumbor/0.9.1

- Fixes issue with redis storage.

0.9.0 - http://pypi.python.org/pypi/thumbor/0.9.0

- Serious BUG Fix. OpenCV Detector data was being returned incorrectly.

0.8.2 - http://pypi.python.org/pypi/thumbor/0.8.2

- Minor Fixes.
- Performance Fixes.

0.8.0 - http://pypi.python.org/pypi/thumbor/0.8.0

- Ticket #41 - Store in the storage the detection results for later usage.

0.7.14 - http://pypi.python.org/pypi/thumbor/0.7.14

- Minor Fixes.

0.7.11 - http://pypi.python.org/pypi/thumbor/0.7.11

- Loader and file storage fixed.

0.7.10 - http://pypi.python.org/pypi/thumbor/0.7.10

- Fit-in bug fixed.

0.7.9 - http://pypi.python.org/pypi/thumbor/0.7.9

- Some performance fixes and MixedStorage.

0.7.8 - http://pypi.python.org/pypi/thumbor/0.7.8

- Ticket #36 - Change Mongo Storage to use GridFS

0.7.7 - http://pypi.python.org/pypi/thumbor/0.7.7

- Ticket #29 - Create an OpenCV Engine

0.7.6 - http://pypi.python.org/pypi/thumbor/0.7.6

- Ticket #35 - MySQL Storage
- Ticket #31 - NoStorage Storage needs to be updated to include no crypto support

0.7.5 - http://pypi.python.org/pypi/thumbor/0.7.5

- Ticket #34 - Meta should have the option of returning as jsonp (REOPENED)

0.7.4 - http://pypi.python.org/pypi/thumbor/0.7.4

- Ticket #34 - Meta should have the option of returning as jsonp

0.7.2 - http://pypi.python.org/pypi/thumbor/0.7.2

- Ticket #32 - Allow unlimited dimensions of images

0.7.0 - http://pypi.python.org/pypi/thumbor/0.7.0

- Ticket #30 - Allow users to use a fit-in flag

0.6.5 - http://pypi.python.org/pypi/thumbor/0.6.5

- Ticket #16 - NoStorage Storage
- Ticket #24 - OpenCV File Issue
- Ticket #26 - BUG: Redis Configuration does not work
- Ticket #27 - BUG: Issue with cropping

0.6.4 - http://pypi.python.org/pypi/thumbor/0.6.4

- Ticket #24 - OpenCV File Issue

0.6.3 - http://pypi.python.org/pypi/thumbor/0.6.3

- Some refactoring and added App and Handler inheritance support.

0.6.2 - http://pypi.python.org/pypi/thumbor/0.6.2

- Ticket #7 - Validate for file size on the http loader

0.6.1 - http://pypi.python.org/pypi/thumbor/0.6.1

- Switched encryption from Triple-Des to AES due to standardization between programming languages.

0.5.1 - http://pypi.python.org/pypi/thumbor/0.5.1

- Fixed a bug with encrypting relative dimension images.

0.5.0 - http://pypi.python.org/pypi/thumbor/0.5.0

- Ticket #5 - Switch the unencrypted URL to be /unsafe and the encrypted to be the default

0.4.1 - http://pypi.python.org/pypi/thumbor/0.4.1

- Ticket #4 - Bug in the encrypted URL generation and parsing

0.4.0 - http://pypi.python.org/pypi/thumbor/0.4.0

- Ticket #2 - Command-line application to generate urls

0.3.0 - http://pypi.python.org/pypi/thumbor/0.3.0

- Ticket #1 - URL Cryptography Support (FIXED)
- Internal logic refactored.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index