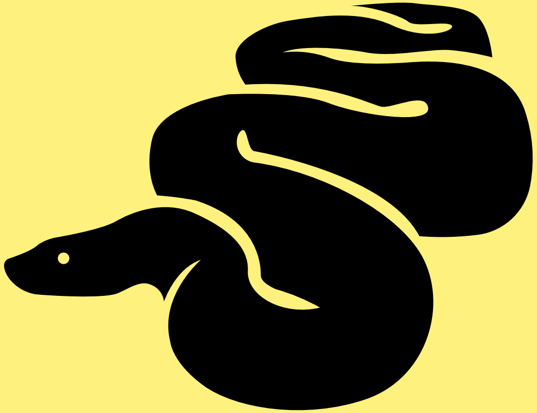
Image processing in 2017



Alexander Karpinsky
Uploadcare.com

About myself

Pillow core team member.

Maker of the Pillow-SIMD library.

What I do

On-the-fly image processing service in Uploadcare.

- High performance
- Reliability
- Scalability
- Built on top of Pillow-SIMD

Libraries

Pillow

- PIL fork (Python Imaging Library). Founded in 1995
- Native extension for Python
- Supported versions: 2.7, 3.3+, pypy, pypy3

python-pillow.org

Pillow-SIMD

- Since May 2016
- Drop-in replacement for Pillow
- Instruction sets: SSE4 (by default), AVX2

github.com/uploadcare/pillow-simd

OpenCV

- Stands for Open Computer Vision. Founded in 2000
- Includes a popular Python binding
- Supported versions: 2.7, 3.4+. No pypy support

opencv.org

VIPS

- Founded in 1993, ahead of its time
- The "pyvips" binding is supported by the author
- Supported versions: 2.7, 3.3+, pypy, pypy3

jcupitt.github.io/libvips/

ImageMagick & GraphicsMagick

- Well-known libraries. Founded in 1990
- The "Wand" binding is based on ctypes and looks abandoned
- The "pgmagick" binding is based on Boost.Python. No pypy support

imagemagick.org, graphicsmagick.org

Performance

Always check your output

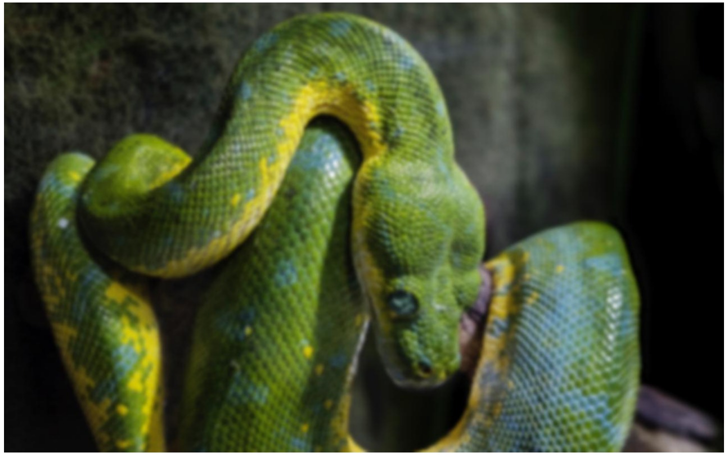
```
from PIL import Image, ImageFilter.BoxBlur
im.filter(ImageFilter.BoxBlur(3))
...
```

```
import cv2
cv2.blur(im, ksize=(3, 3))
...
```

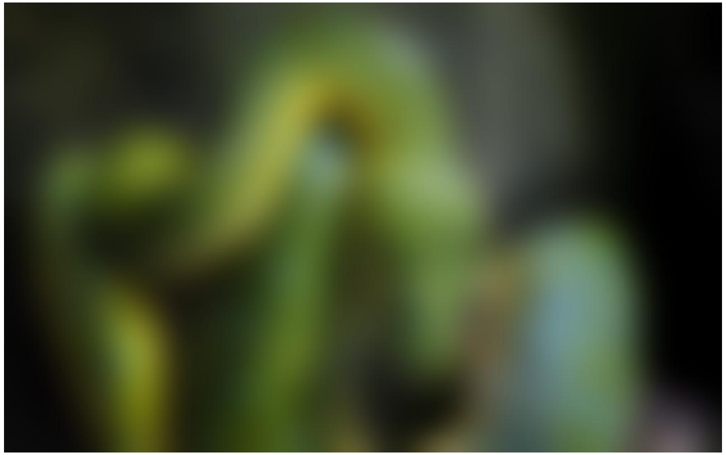
The problem

```
cv2.GaussianBlur(im, (window, window), radius)
```

`radius = 3` 58 ms



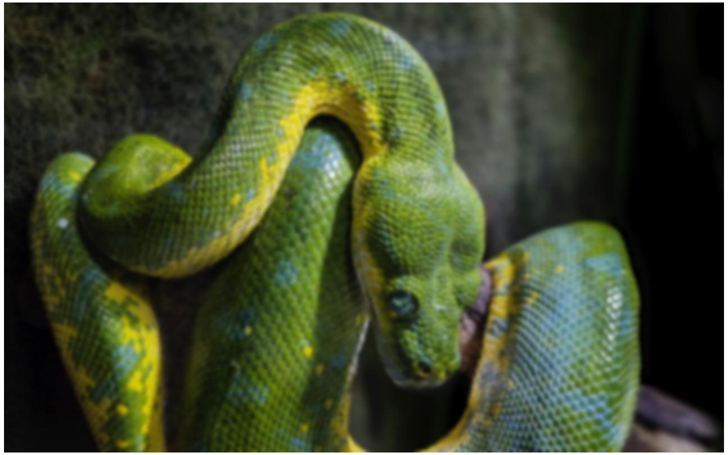
`radius = 30` 880 ms



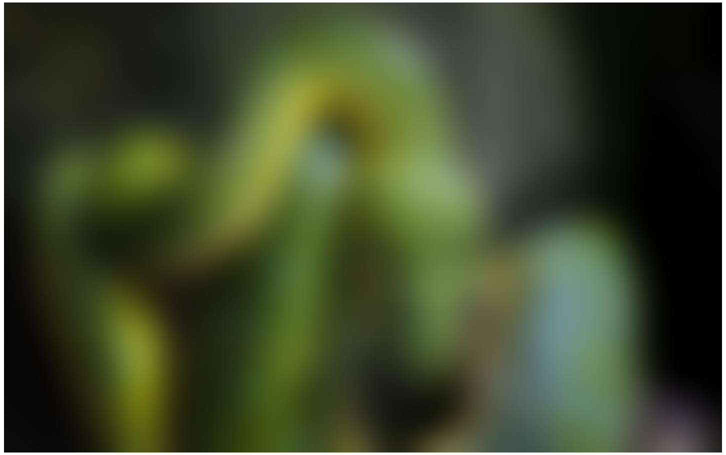
The problem

```
im.filter(ImageFilter.GaussianBlur(radius))
```

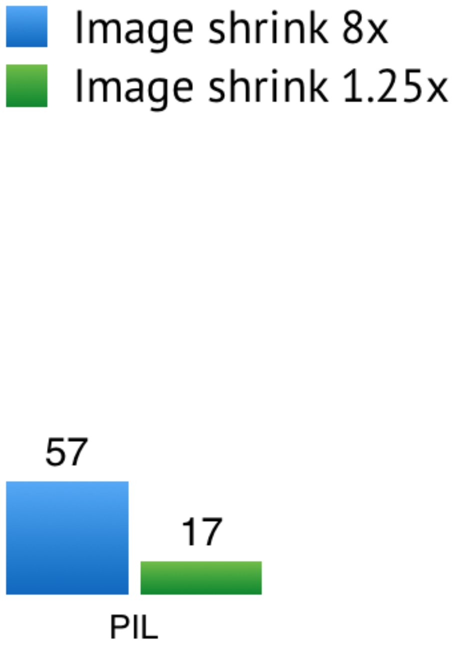
`radius = 3` 60 ms



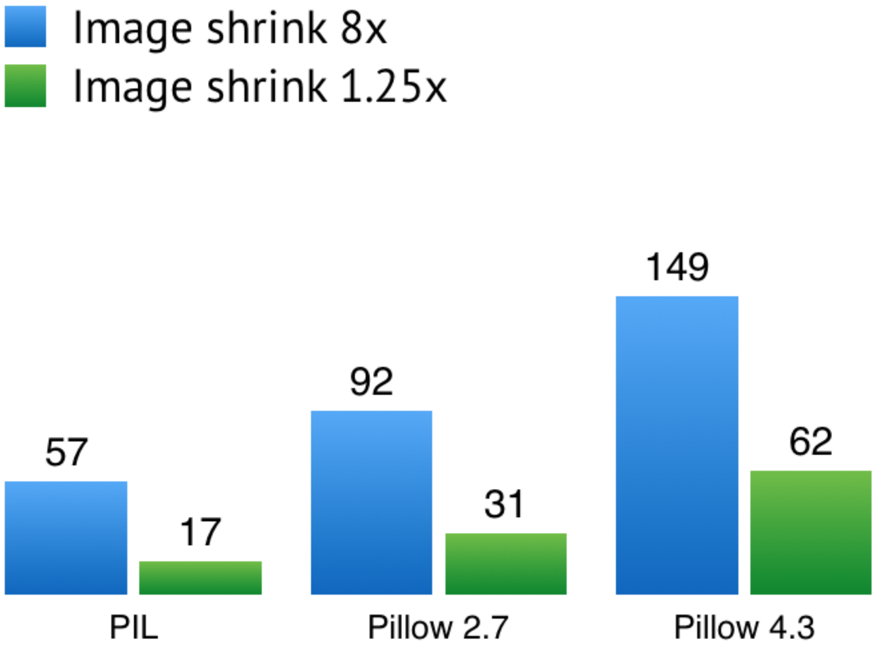
`radius = 30` 61 ms



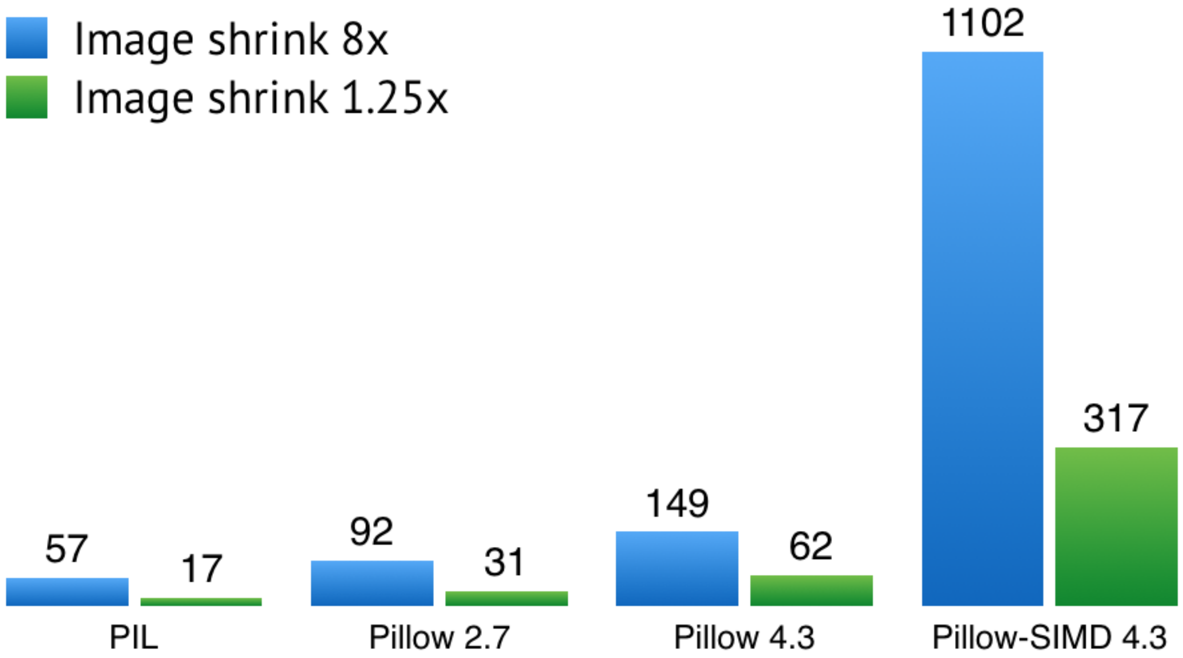
Resampling speed in Pillow, Mpx/s



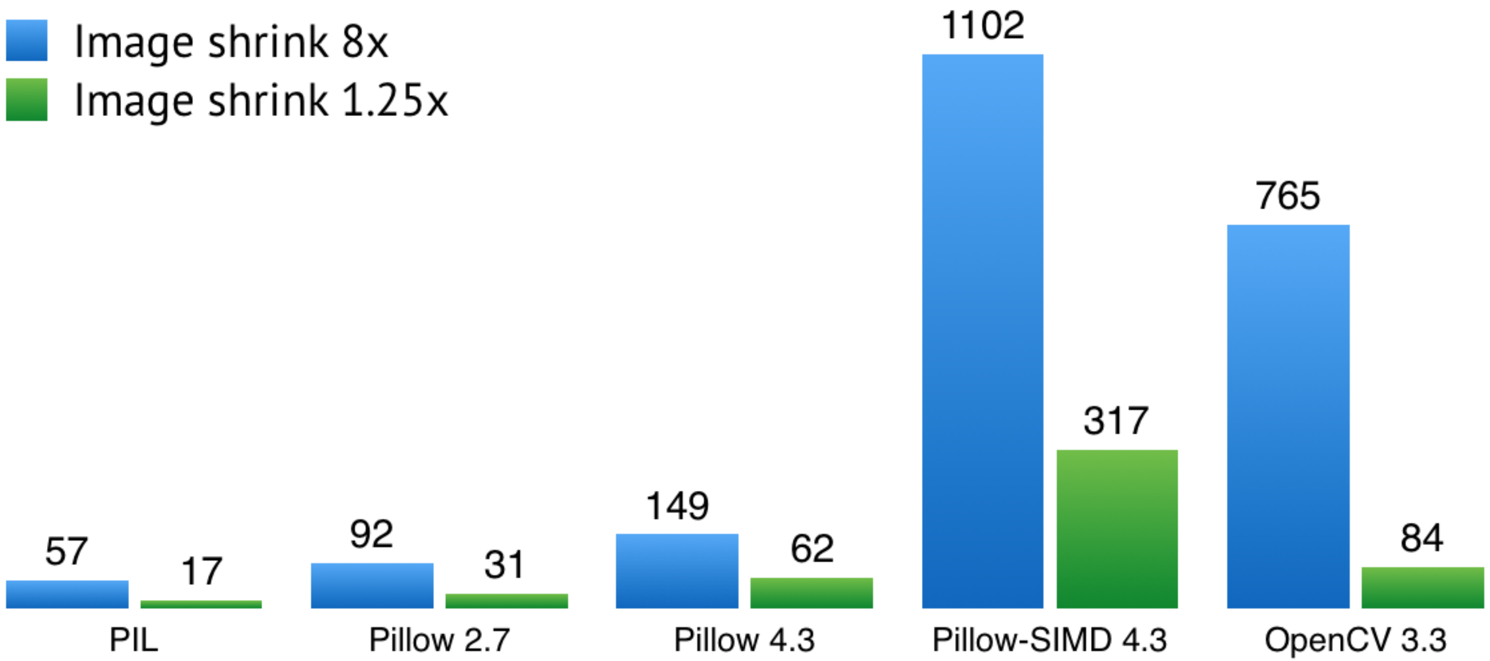
Resampling speed in Pillow, Mpx/s



Resampling speed in Pillow, Mpx/s



Resampling speed in Pillow, Mpx/s

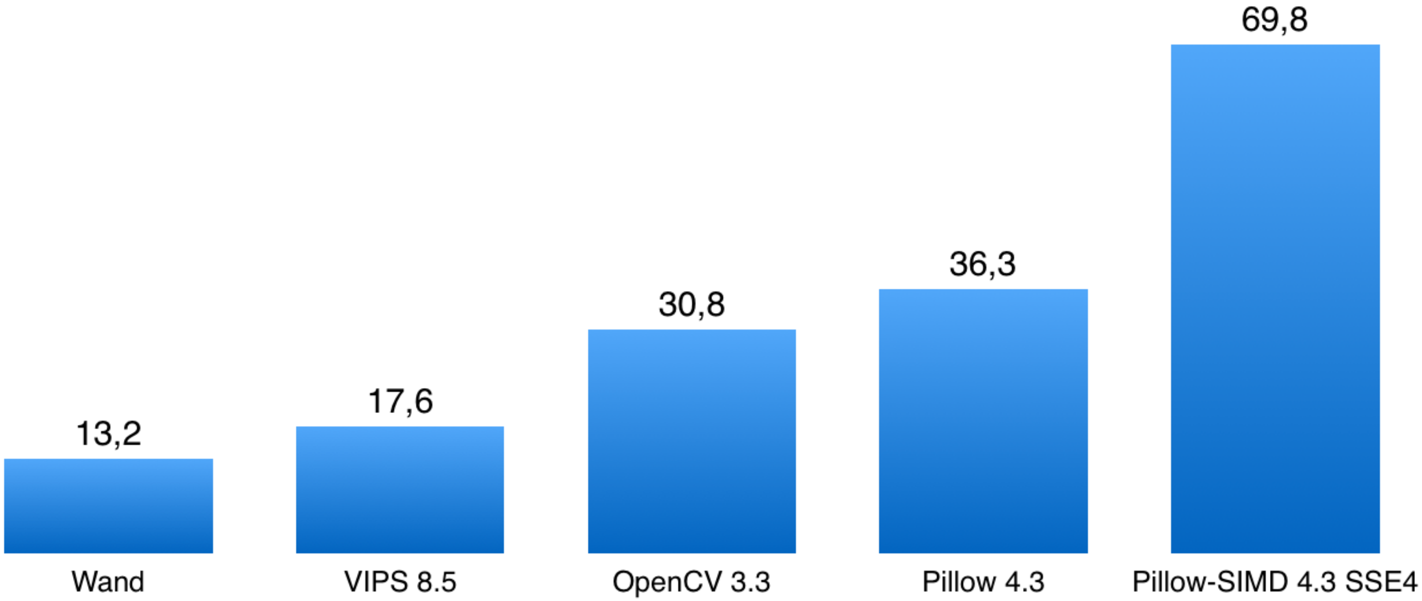


Pillow-SIMD speeds up

- Resampling: 4 – 7 times
- Gaussian blur: 2.8 times
- Kernel filter 3×3 or 5×5: 11 times
- Multiplication and division by alpha channel: 4 and 10 times
- Alpha compositing: 5 times
- And counting…

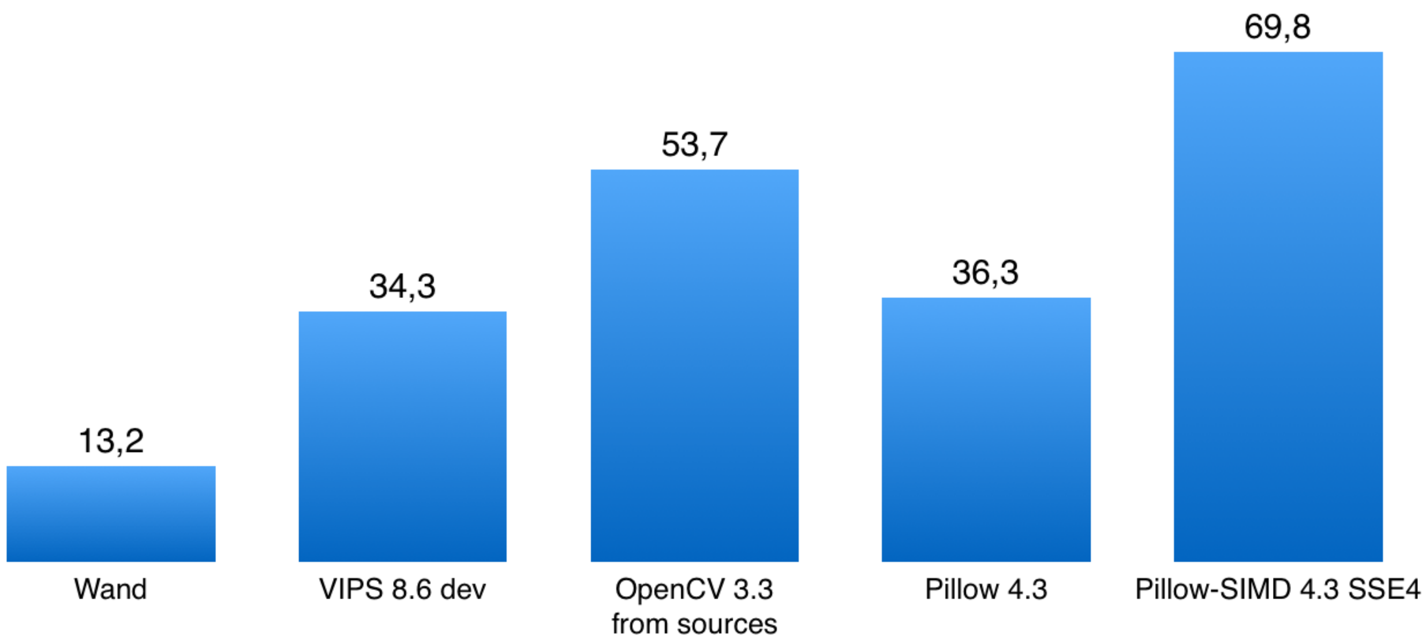
Some sequence of operations, Mpx/s

Load, rotate by 90°, reduce 2.5 times, apply blur, save to JPEG.



Some sequence of operations, Mpx/s

Results when you invest some time.



Benchmarking framework

Results page

https://python-pillow.org/pillow-perf/

Benchmark sources

https://github.com/python-pillow/pillow-perf

Concurrent working

Performance metrics

- **Actual execution time for one operation execution**

  Important on desktops.

- **Operations flow throughput**

  Becomes more important on servers.

Concurrent working levels

1. Application level

   Actual execution time doesn't change.

   Throughput grows in proportion to the number of cores.

Concurrent working levels

2. Graphical operation level

Actual execution time lowers.

Throughput grows **not** in proportion to the number of cores.

Concurrent working levels

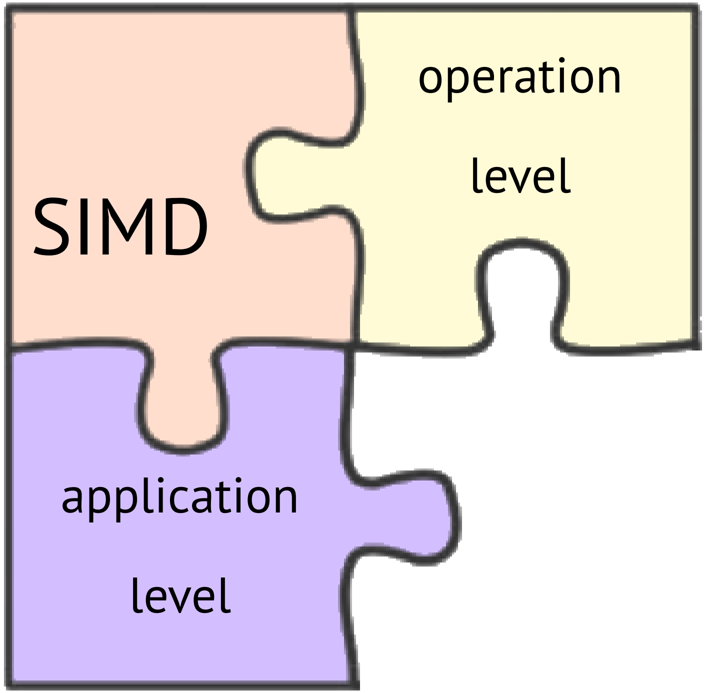
3. Data and CPU instructions level (SIMD)

Actual execution time lowers.

Throughput grows.

Win-win.

Combining methods



SIMD

operation level

application level

Multithreading

Release GIL

Pillow, OpenCV, pyvips, Wand

Doesn't release

pgmagick

The N + 1 rule

Create not more than N + 1 workers,

where N is a number of CPU cores or threads.

Worker – a process or thread doing the processing.

Asynchronous work

Executing imaging operations blocks event loop,

even if a library releases GIL.

```
@gen.coroutine
def get(self, *args, **kwargs):
    im = process_image(...)
    ...
```

Asynchronous work

```
@run_on_executor(executor=ThreadPoolExecutor(1))
def process_image(self, ...):
    ...
@gen.coroutine
def get(self, *args, **kwargs):
    im = yield process_image(...)
    ...
```

File input/output

Lazy loading

```
>>> from PIL import Image
>>> %time im = Image.open('cover.jpg')
Wall time: 1.2 ms
>>> im.mode, im.size
('RGB', (2152, 1345))
```

Lazy loading

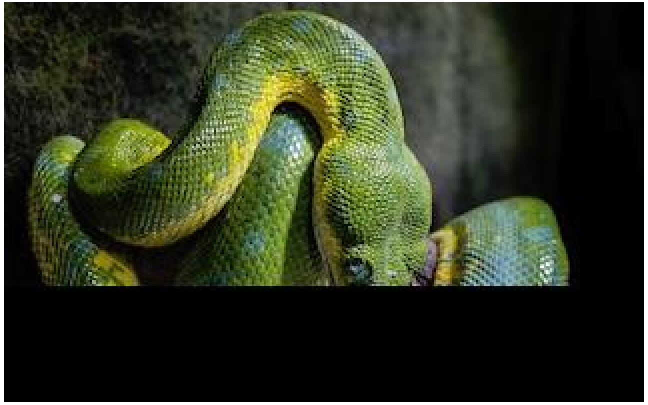
```
>>> from PIL import Image
>>> %time im = Image.open('cover.jpg')
Wall time: 1.2 ms
>>> im.mode, im.size
('RGB', (2152, 1345))
>>> %time im.load()
Wall time: 73.6 ms
```

Broken images mode

```
from PIL import Image
Image.open('trucated.jpg').save('trucated.out.jpg')
IOError: image file is truncated (143 bytes not processed)
```

Broken images mode

```
from PIL import Image, ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True
Image.open('trucated.jpg').save('trucated.out.jpg')
```



	Pillow	VIPS	OpenCV	IM
Number of codecs	17	12+	8	66
Broken images	✅	✅	✅	✅
Lazy loading	✅	✅	❌	❌
Reading EXIF and ICC	✅	✅	❌	✅
Auto rotation based on EXIF	❌	✅	✅	✅

OpenCV quirks

```
cv2.imread(filename)
```

- Auto rotates JPEG files based on EXIF
- Ignores alpha channel in PNG files

OpenCV quirks

```
cv2.imread(filename, flags=cv2.IMREAD_UNCHANGED)
```

- Preserves alpha channel in PNG files
- Stops EXIF-based autorotation

OpenCV, why?

- Few codecs
- No lazy loading
- No access to EXIF and ICC
- Odd flags

OpenCV is not designed to work with untrusted sources.

Solution

Solution

OpenCV images are numpy arrays.

```
import numpy
from PIL import Image
...
pillow_image = Image.open(filename)
cv_image = numpy.array(pillow_image)
```

Solution

```
import numpy
from PIL import Image
...
pillow_image = Image.fromarray(cv_image, "RGB")
pillow_image.save(filename)
```

Questions

Slides: homm.github.io/image-libs-2017/

Email: ak@uploadcare.com