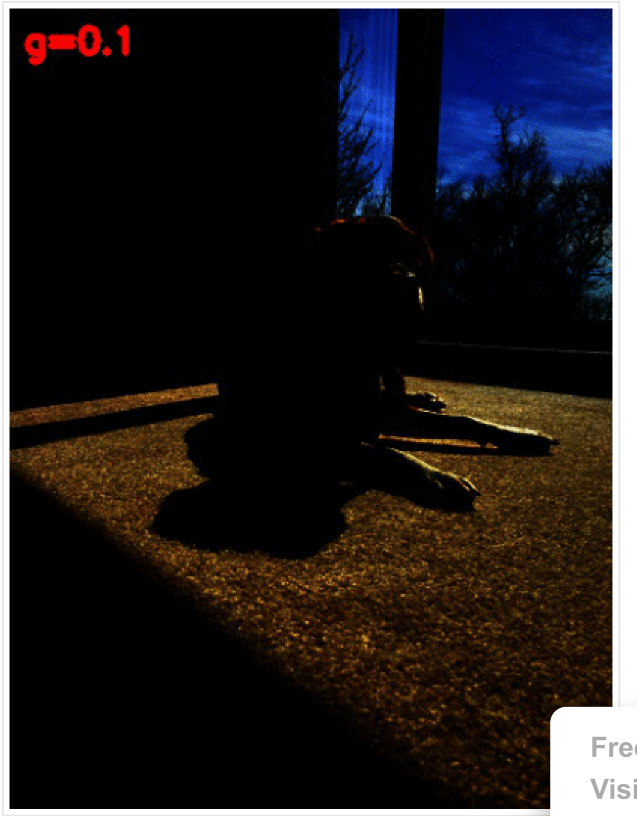
Navigation

pyimagesearch
be awesome at building image search engines

OpenCV Gamma Correction

by **Adrian Rosebrock** on October 5, 2015 in **Image Processing**, **Tutorials**



Free 17-day crash course on Computer Vision, OpenCV, and Deep Learning

Did you know that the human eye perceives color and luminance differently than the sensor on your smartphone or digital camera?

You see, when *twice* the number of photons hit the sensor of a digital camera, it receives *twice* the signal (a linear relationship). However, that's not how our human eyes work. Instead, we perceive *double the amount of light* as only a *fraction* brighter (a non-linear relationship)! Furthermore, our eyes are also much more sensitive to changes in dark tones than brighter tones (another non-linear relationship).

In order to account for this we can apply *gamma correction*, a translation between the sensitivity of our eyes and sensors of a camera.

Looking for the source code to this post?
Jump right to the downloads section.

In the remainder of this post I'll demonstrate how you can implement a super fast, dead-s d OpenCV.

Gamma correction and the Power Law Transform

Gamma correction is also known as the *Power Law Transform.* First, our image pixel inte *0, 1.0]*. From there, we obtain our output gamma corrected image by applying the following

O = I ^ (1 / G)

Where *I* is our input image and *G* is our gamma value. The output image *O* is then scalec

Gamma values < *1* will shift the image towards the darker end of the spectrum while gam . A gamma value of *G=1* will have no affect on the input image:

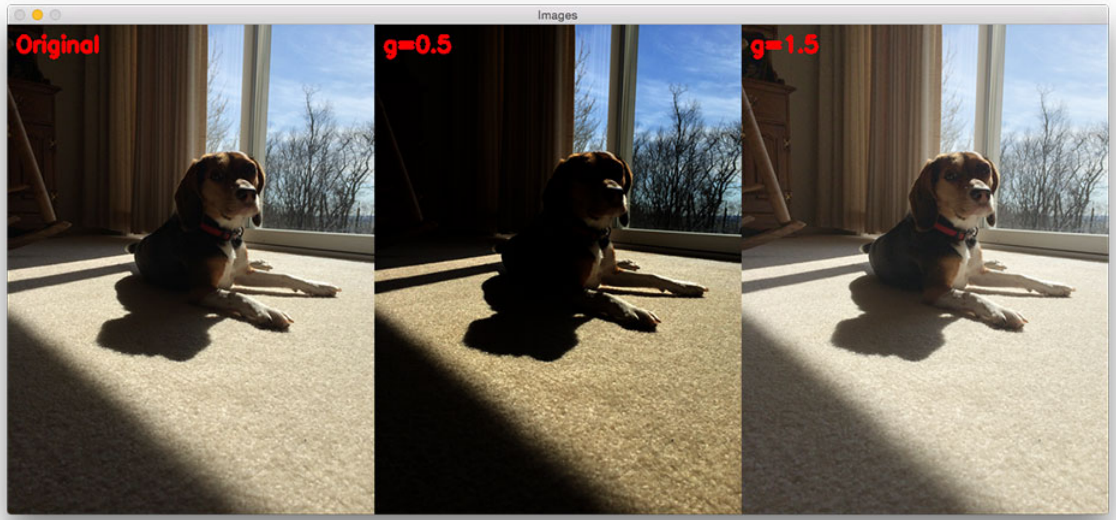


Figure 1: Our original image *(left)*; Gamma correction with *G < 1 (center)*, notice how the gamma adjusted image is much darker than the original image; Gamma correction with *G > 1 (right)*, this time the output image is much lighter than the original.

OpenCV Gamma Correction

Now that we understand what gamma correction is, let's use OpenCV and Python to implement it. Open up a new file, name it `adjust_gamma.py` , and we'll get started:

OpenCV Gamma Correction Python

```python
# import the necessary packages
from __future__ import print_function
import numpy as np
import argparse
import cv2

def adjust_gamma(image, gamma=1.0):
    # build a lookup table mapping the pixel values [0, 255] to
    # their adjusted gamma values
    invGamma = 1.0 / gamma
    table = np.array([((i / 255.0) ** invGamma) * 255
        for i in np.arange(0, 256)]).astype("uint8")

    # apply gamma correction using the lookup table
```

```
15      return cv2.LUT(image, table)
```

Lines 2-5 simply import our necessary packages, nothing special here.

We define our `adjust_gamma` function on **Line 7**. This method requires a single parameter, `image` , which is the image we want to apply gamma correction to. A second (optional) value is our `gamma` value. In this case, we default `gamma=1.0` , but you should supply whatever value is necessary to obtain a decent looking corrected image.

There are two (easy) ways to apply gamma correction using OpenCV and Python. The first method is to simply leverage the fact that Python + OpenCV represents images as NumPy arrays. All we need to do is scale the pixel intensities to the range *[0, 1.0]*, apply the transform, and then scale back to the range *[0, 255]*. Overall, the NumPy approach involves a division, raising to a power, followed by a multiplication — this tends to be very fast since all these operations are vectorized.

However, there is an even *faster* way to perform gamma correction thanks to OpenCV. All we need to do is build a table (i.e. dictionary) that maps the *input pixel values* to the *output gamma corrected values*. OpenCV can then take this table and quickly determine the output value for a given pixel in *O(1)* time.

For example, here is an example lookup table for `gamma=1.2` :

```
OpenCV Gamma Correction                                                    Shell
0 => 0
1 => 2
2 => 4
3 => 6
4 => 7
5 => 9
6 => 11
7 => 12
8 => 14
9 => 15
10 => 17
```

The *left column* is the **input pixel value** while the *right column* is the **output pixel value** after applying the power law transform.

Lines 11 and 12 build this lookup table by looping over all pixel values in the range *[0, 255]*. The pixel value is then scaled to the range *[0, 1.0]* followed by being raised to the power of the inverse gamma — this value is then stored in the `table` .

Lastly, all we need to do is apply the `cv2.LUT` function (**Line 15**) to take the input `image` and the `table` and find the correct mappings for each pixel value — it's a simple (and yet very speedy) operation!

Let’s continue on with our example:

OpenCV Gamma Correction Python

```
17 # construct the argument parse and parse the arguments
18 ap = argparse.ArgumentParser()
19 ap.add_argument("-i", "--image", required=True,
20     help="path to input image")
21 args = vars(ap.parse_args())
22
23 # load the original image
24 original = cv2.imread(args["image"])
```

Lines 17-21 handle parsing command line arguments. We only need a single switch here, `--image` , which is the path to where our input image resides on disk. **Line 24** takes the path to our image and loads it.

Let’s explore gamma correction by using a variety of gamma values and inspecting the output image for each:

OpenCV Gamma Correction Python

```
26 # loop over various values of gamma
27 for gamma in np.arange(0.0, 3.5, 0.5):
28     # ignore when gamma is 1 (there will be no change to the image)
29     if gamma == 1:
30         continue
31
32     # apply gamma correction and show the images
33     gamma = gamma if gamma > 0 else 0.1
34     adjusted = adjust_gamma(original, gamma=gamma)
35     cv2.putText(adjusted, "g={}".format(gamma), (10, 30),
36         cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 255), 3)
37     cv2.imshow("Images", np.hstack([original, adjusted]))
38     cv2.waitKey(0)
```

On **Line 27** we start by looping over `gamma` values in the range *[0, 3.0]* (the `np.arange` function is *non-inclusive*), incrementing by *0.5* at each step.

In the case that our `gamma` value is *1.0*, we simply ignore it (**Lines 29 and 30**) since `gamma=1.0` will not change our input image.

From there, **Lines 33-38** apply gamma correction to our image and display the output result.

To see gamma correction in action, just open up a terminal and execute the following command:

OpenCV Gamma Correction Shell

```
$ python adjust_gamma.py --image example_01.png
```

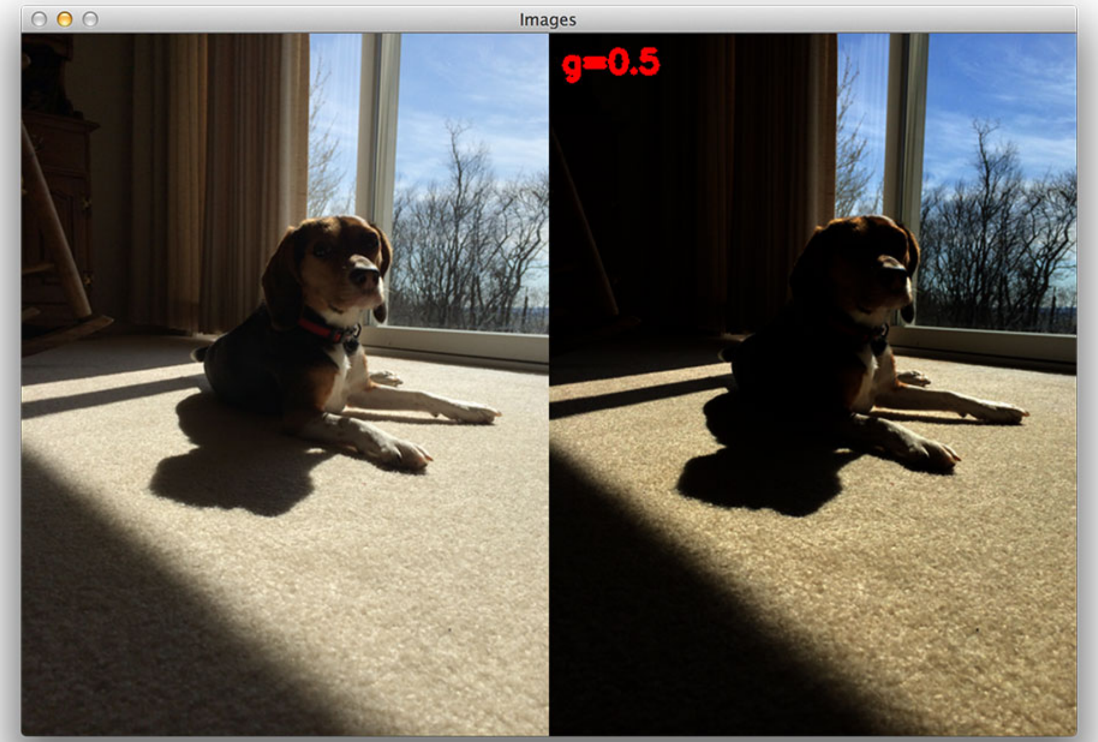


Figure 2: When applying gamma correction with *G < 1*, the output image is will darker than the original input image.

Notice for `gamma=0.5` that the gamma corrected image *(right)* is substantially darker than the input image *(left)* which is already quite dark — we can barely see any detail on the dog's face in the original yet, let alone the gamma corrected version!

However, at `gamma=1.5` the image starts to lighten up and we can see more detail:

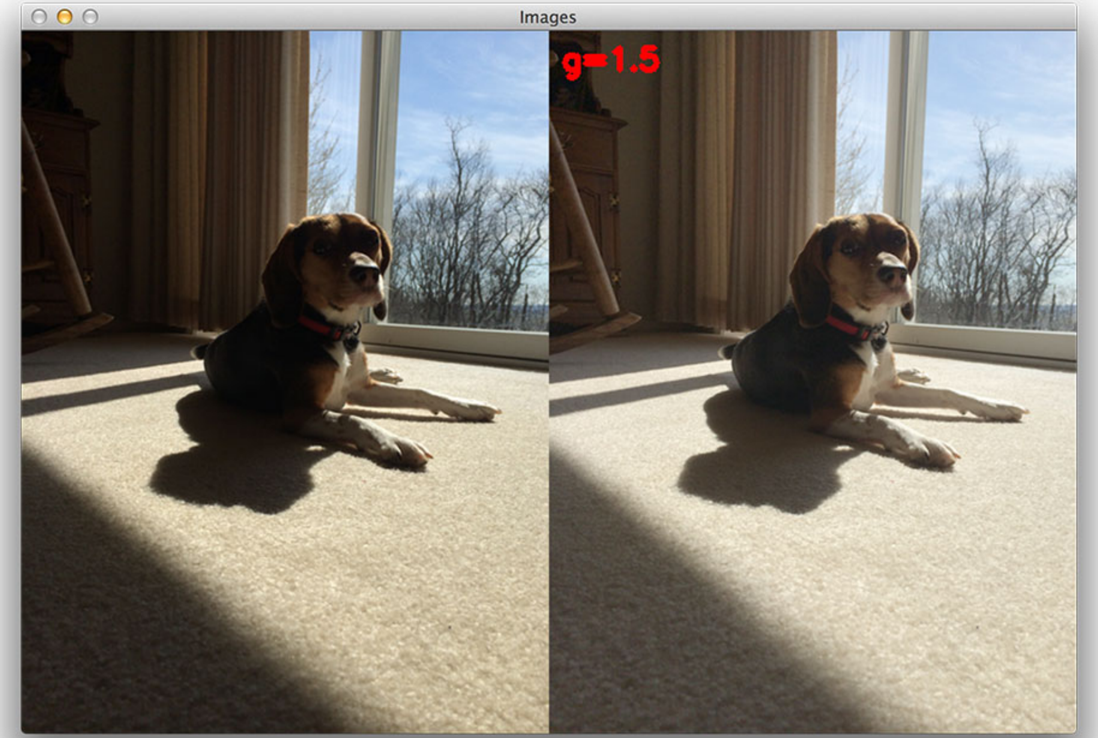


Figure 3: As the gamma value reaches 1.0 and starts to exceed it, the image lightens up and we can see more detail.

By the time we reach `gamma=2.0` , the details in the image are fully visible.

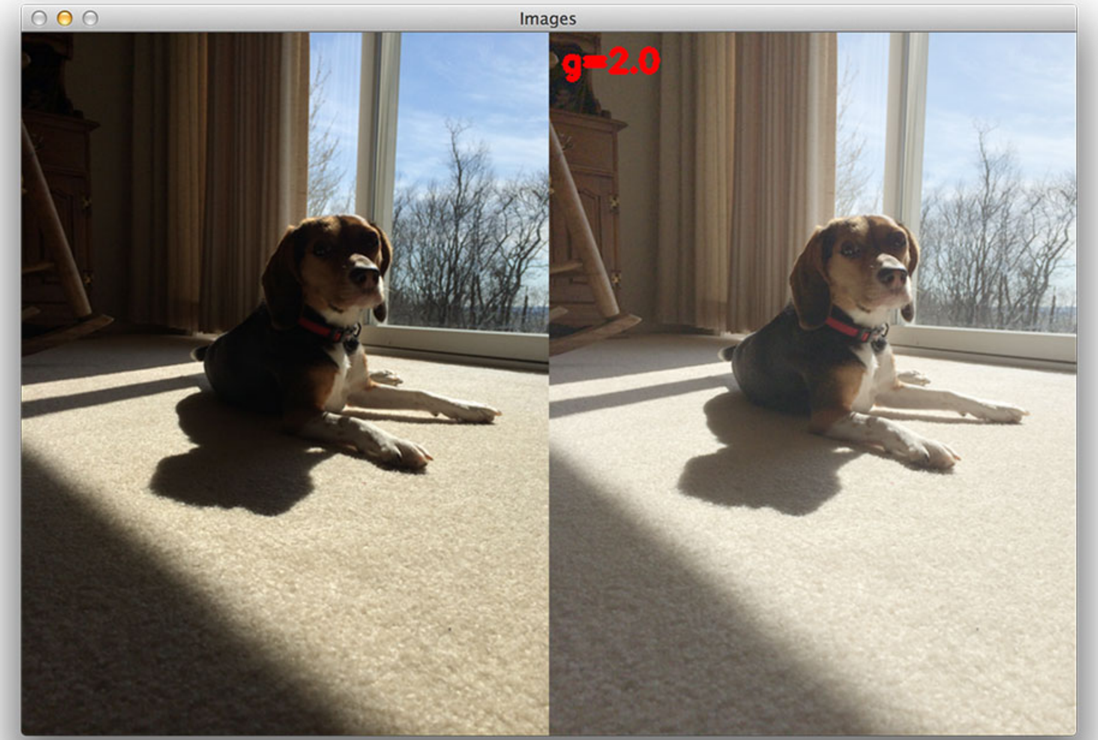


Figure 4: Now at *gamma=2.0*, we can fully see the details on the dogs face.

Although at `gamma=2.5` , the image starts to appear “washed out”:

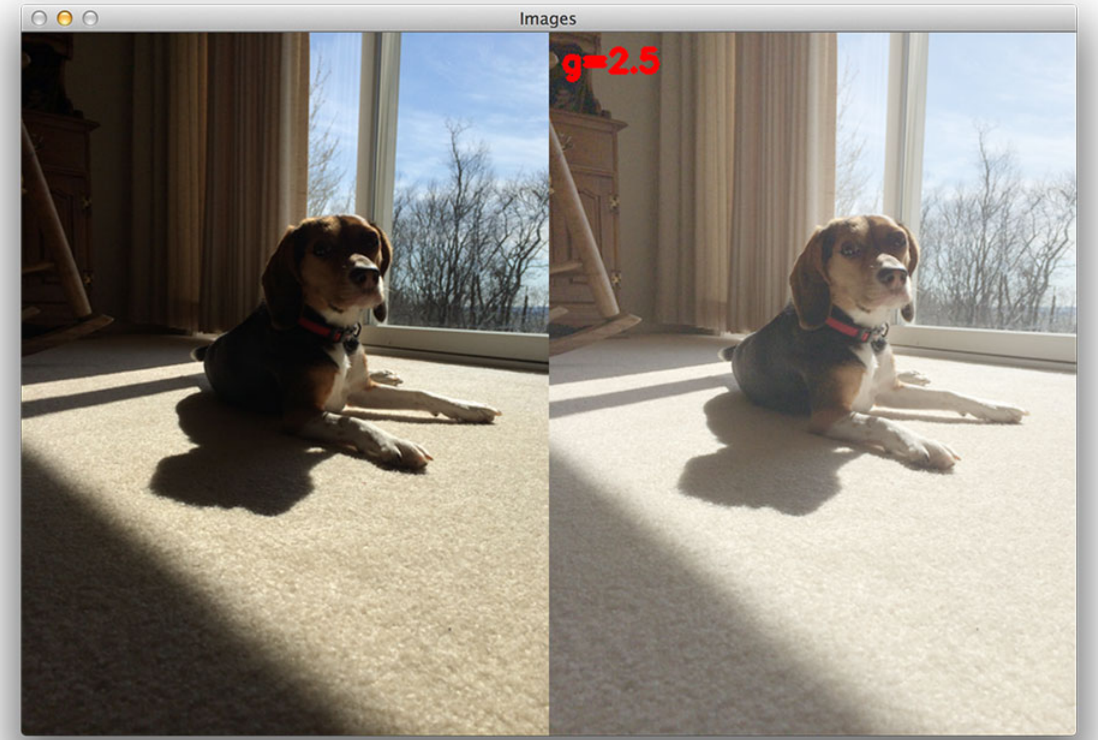


Figure 5: However, if we carry gamma correction too far, the image will start to appear washed out.

Let's give another image a try:

OpenCV Gamma Correction Shell
```
$ python adjust_gamma.py --image example_02.png
```

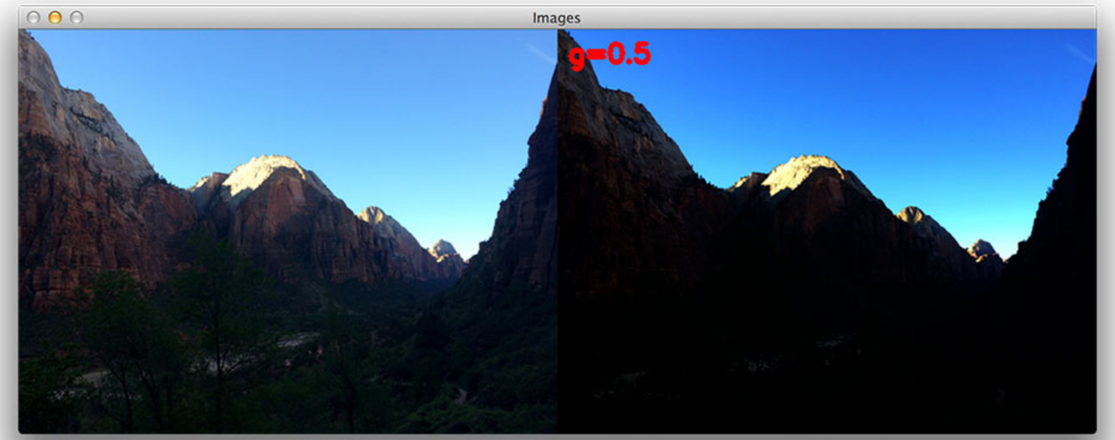


Figure 6: After applying gamma correction with *gamma=0.5*, we cannot see any detail in this image.

Just like in `example_01.png` , a gamma value of *0.5* makes the input image appear darker than it already is. We can't really make out any detail in this image, other than there is sky and what appears to be a mountain range.

However, this changes when we apply gamma correction with `gamma=1.5` :

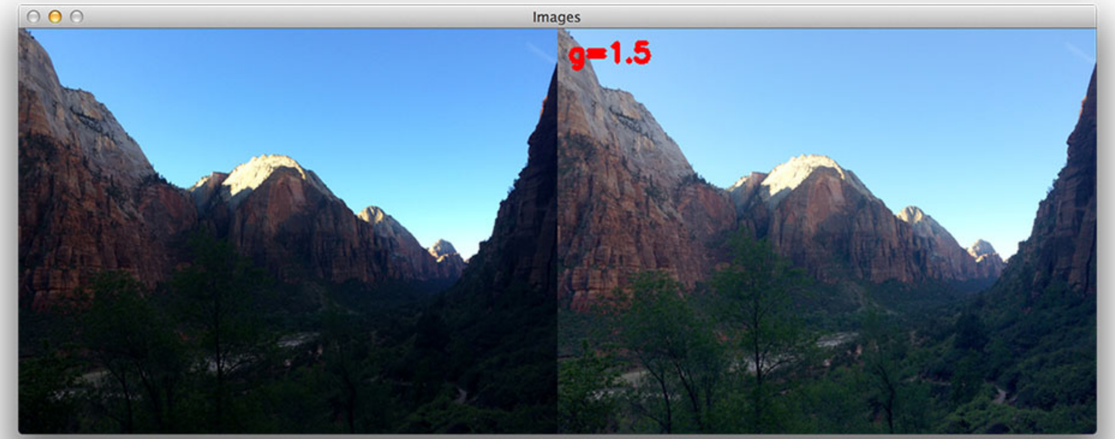


Figure 7: Optimal results are obtained near *gamma=1.5*.

Now we can see that the image has become much lighter — we can even start to see there are trees in the foreground, something that is not entirely apparent from the original input image on the *left*.

At `gamma=2.0` the image starts to appear washed out, but again, the difference between the original image and the gamma corrected image is quite substantial:

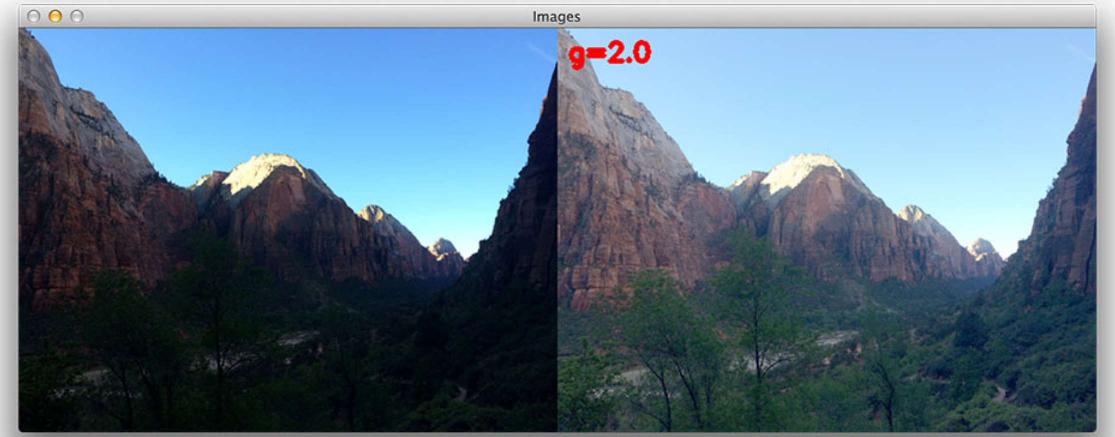


Figure 8: But again, we can carry gamma correction too far and washout our image.

Summary

In this blog post we learned about *gamma correction*, also called the *Power Law Transform.* We then implemented gamma correction using Python and OpenCV.

The reason we apply gamma correction is because our eyes perceive color and luminance differently than the sensors in a digital camera. When a sensor on a digital camera picks up *twice* the amount of photons, the signal is *doubled*. However, our eyes do not work like this. Instead, our eyes perceive *double* the amount of light as only a *fraction* brighter. Thus, while a digital camera has a *linear* relationship between brightness our eyes have a *non-linear* relationship. In order to account for this relationship we apply gamma correction.

Be sure to download the code to this post and try applying gamma correction to your own photos. Try to go through your photo collection and find images that are either *excessively dark* or *very bright and washed out*. Then perform gamma correction on these images and see if they become more visually appealing.

Downloads:

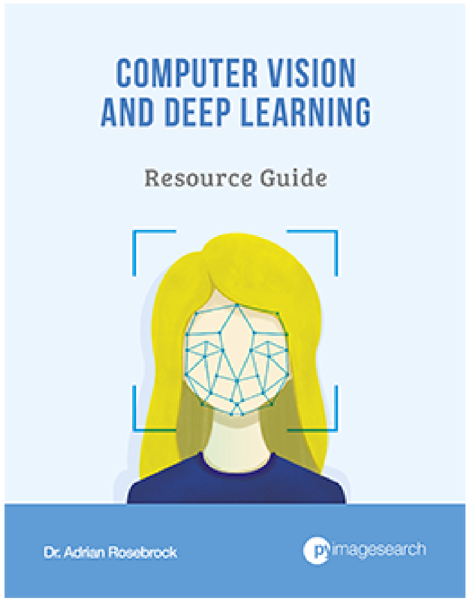
If you would like to download the code and images used in this post, please enter your email address in the form below. Not only will you get a .zip of the code, I'll also send you a **FREE 17-page Resource Guide on Computer Vision, OpenCV, and Deep Learning.** Inside you'll find my hand-picked tutorials, books, courses, and libraries to help you master CV and DL! Sound good? If so, enter your email address and I'll send you the code immediately!

Email address:

Your email address

DOWNLOAD THE CODE!

Resource Guide (it's totally free).



Enter your email address below to get my **free 17-page Computer Vision, OpenCV, and Deep Learning Resource Guide PDF**. Inside you'll find my hand-picked tutorials, books, courses, and Python libraries to help you master computer vision and deep learning!

Your email address

DOWNLOAD THE GUIDE!

filter, **gamma correction**, **image processing**

< Implementing the Max RGB filter in OpenCV

Scraping images with Python and Scrapy >

36 Responses to *OpenCV Gamma Correction*

Abhinav October 16, 2015 at 2:33 am # REPLY

Hello Adrian,
Your Blog posts has got me very interested in OpenCV. Keep posting!
However, I think there is an mistake in the above code, Line 11 and 12 are in each other's place.

Adrian Rosebrock October 16, 2015 at 6:15 am # REPLY

Hey Abhinav — Lines 12 and 12 are actually a single statement that wrap two lines of code.

Brian November 17, 2015 at 6:09 pm # REPLY

Is there any way to perform localized gamma correction? For example, in the picture with dog, is there a function that would gamma correct (lighten) the areas that are in the shadows but would essentially ignore the areas that are in the sun since they are already bright?

This might help to "equalize" the picture (for lack of the correct word). Once the picture is equalized, other OpenCV functions could the be used to detect things without having to worry about bright and dark regions negatively impacting the results. What do you think?

Adrian Rosebrock November 18, 2015 at 6:29 am # REPLY

Hey Brian — localized Gamma correction is an active area of research. I know there was a paper written about it a few years ago. Otherwise, OpenCV does implement Contrast Limited Adaptive Histogram Equalization (CLAHE) which helps in adjusting local areas of contrast.

Rossi September 7, 2016 at 3:06 pm # REPLY

Hello Adrian.
First of all, I'd like to thank you for all support that your blog is giving to me. Thus, I'd like to know if you have some explanation about image filters, like the Difference of Gaussians and Histogram Equalization.
Thank you again.
Best Regards

Adrian Rosebrock September 8, 2016 at 1:21 pm # REPLY

Hey Rossi, thanks for the comment. I discuss Difference of Gaussians and Histogram Equalization in quite a bit of detail inside the PyImageSearch Gurus course. I would suggest starting there.

Rossi September 8, 2016 at 4:57 pm # REPLY

Thank you Adrian.

YuriiChernyshov November 27, 2016 at 2:07 pm # REPLY

Hi Adrian,
Consequent question 🙂
How to do opposite? How to know whether it is necessary to increase or decrease level of gamma? How to know a level of gamma?

Adrian Rosebrock November 28, 2016 at 10:24 am # REPLY

In this case you normally wouldn't unless you have calibrated your camera using a color filter. Gamma correction is almost always a post-processing step after the image has already been captured.

YuriiChernyshov December 2, 2016 at 8:12 pm # REPLY

Sorry, I was not clear with my question.
I was talking about some functionality which normalize gamma before pass image for further processing. In your solution you show how to change gamma assuming that client already knows what to do. But in order to know what to do (increase or decrease) it is necessary to analyze image somehow … The question is "how" ?

Adrian Rosebrock December 5, 2016 at 1:37 pm # REPLY

You would need to visually inspect the image. This is how graphic designers and photographers edit an image. They capture an image, open it in Photoshop (or whatever other photo editing software) and adjust any gamma levels they see fit.

If you were deploying a computer vision product you would inspect the environment and captured images, then build in the gamma correction to the program.

Yurvan Igo May 18, 2017 at 12:51 am # REPLY

How to save the new image has changed

Adrian Rosebrock May 18, 2017 at 11:50 am # REPLY

You can use the `cv2.imwrite` function to write an image to file:

```
cv2.imwrite("foo.png", image)
```

You can learn more about the basics of OpenCV and computer vision inside Practical Python and OpenCV.

Irene Among May 25, 2017 at 11:12 pm # REPLY

Hi, I like your explanations very clear.
I am doing projects in opencv and python but totally new to it.
Your posts are really helpful.
Thank you.

Adrian Rosebrock May 28, 2017 at 1:15 am # REPLY

Thanks Irene — I'm happy to hear you've found the PyImageSearch blog helpful 🙂 If you are new to OpenCV and computer vision, be sure to take a look at Practical Python and OpenCV. This book is designed to help beginners learn the fundamentals of OpenCV quickly.

ambika September 14, 2017 at 6:49 am # REPLY

Hi,

Great Post! But when I try to save the Gamma corrected output of the image, it saves both images together in one image just as it is shown above. What changes should I make in the code to display as well as save only the output image?

Adrian Rosebrock September 14, 2017 at 7:00 am # REPLY

Remove the `np.hstack` call:

```
cv2.imwrite("output.png", adjusted)
```

Kinvert February 23, 2018 at 9:50 pm # REPLY

Thanks for this! It is helping our students with a project.

Do you have any recommended articles on color correction / white balance?

Students are considering LUT on an RGB / HSV split, then a merge. The tables would have to be a bit different.

Thanks again!

Adrian Rosebrock February 26, 2018 at 2:03 pm # REPLY

I'm glad to hear my tutorials are helping your students, that's great!

I do not have any tutorials on color correction or white balancing. I will try to cover these subjects in the future. The gist is that you need to compute the CDF of both your input and reference images, then transfer the CDF to the input image to correct for the white balance.

esha May 16, 2018 at 5:09 am # REPLY

hello
how i can i apply gamma technique to gray scale filtered image

Adrian Rosebrock May 17, 2018 at 6:57 am # REPLY

You can still use the method in this blog post.

esha May 29, 2018 at 3:39 am # REPLY

hello
the above code is showing error pls help
the error is as follows:
check.py: error: the following arguments are required: -i/–fogimage

Adrian Rosebrock May 31, 2018 at 5:25 am # REPLY

You can resolve this error by reading up on command line arguments.

esha June 4, 2018 at 3:10 am # REPLY

on command line it is showing error
unrecognized argument : example1.png
pls resolve

Adrian Rosebrock June 5, 2018 at 7:55 am # REPLY

I believe your error is due to not utilizing the command line arguments properly. Make sure you read this post.

esha June 4, 2018 at 5:02 am # REPLY

sir i have downloaded ur code nd apply with my images but it is showing error of unrecognized argument

Adrian Rosebrock June 5, 2018 at 7:52 am # REPLY

What is the exact error you are receiving?

esha June 7, 2018 at 1:25 am # REPLY

if possible pls send me code which is applicable to gray images

Dharshan June 29, 2018 at 4:37 am # REPLY

Is there any posts which will analyse the image and set a gamma value based on that,rather than user defining it

Rohil July 6, 2018 at 2:05 am # REPLY

Hi Adrian

I really liked your explanation. My only doubt is why the lookup table method is faster than the first method?

Adrian Rosebrock July 10, 2018 at 8:57 am # REPLY

Which first method are you referring to? As to why a LUT is fast see this sentence:

“OpenCV can then take this table and quickly determine the output value for a given pixel in O(1) time.”

Jibin John July 13, 2018 at 2:02 am # REPLY

Hello Adrian is it a best method for the correction of images taken at night using cameras

Adrian Rosebrock July 13, 2018 at 4:48 am # REPLY

What type of camera are you using? Some cameras include a “smart IR” method that can help perform correction at capture time which would probably be more ideal than trying to perform gamma correction manually.

esha July 13, 2018 at 5:15 am #

REPLY

Is there any posts which will analyse the image and set a gamma value based on that,rather than user defining it

Adrian Rosebrock July 13, 2018 at 5:23 am #

REPLY

No, I do not have any tutorials on automatically setting the gamma value.

anne September 1, 2018 at 3:47 pm #

REPLY

hi
Why did you use inverse gamma instead of applying the normal gamma?

thanks

Leave a Reply

Name (required)

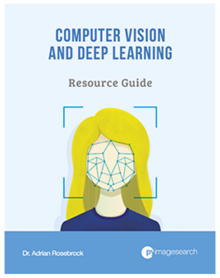
Email (will not be published) (required)

Website

SUBMIT COMMENT

Search...

Resource Guide (it's totally free).



Get your **FREE 17 page Computer Vision, OpenCV, and Deep Learning Resource Guide PDF.** Inside you'll find my hand-picked tutorials, books, courses, and libraries to help you master CV and DL.

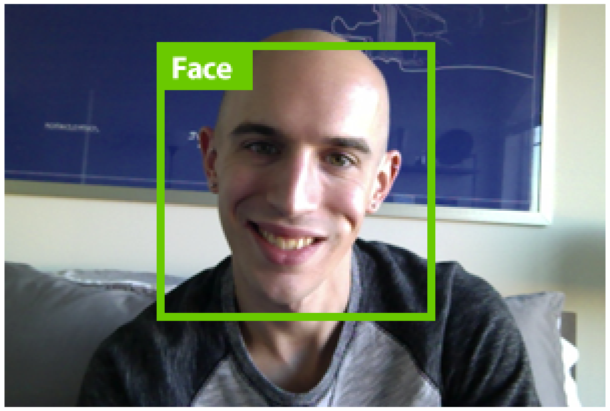
Download for Free!

Deep Learning for Computer Vision with Python Book — OUT NOW!

You're interested in deep learning and computer vision, *but you don't know how to get started.* Let me help. **My new book will teach you all you need to know about deep learning.**

CLICK HERE TO MASTER DEEP LEARNING

You can detect faces in images & video.



Are you interested in **detecting faces in images & video?** But **tired of Googling for tutorials** that *never work?* Then let me help! I guarantee that my new book will turn you into a **face detection ninja** by the end of this weekend. Click here to give it a shot yourself.

CLICK HERE TO MASTER FACE DETECTION

PyImageSearch Gurus: NOW ENROLLING!

The PyImageSearch Gurus course is *now enrolling!* Inside the course you'll learn how to perform:

- Automatic License Plate Recognition (ANPR)
- Deep Learning
- Face Recognition
- *and much more!*

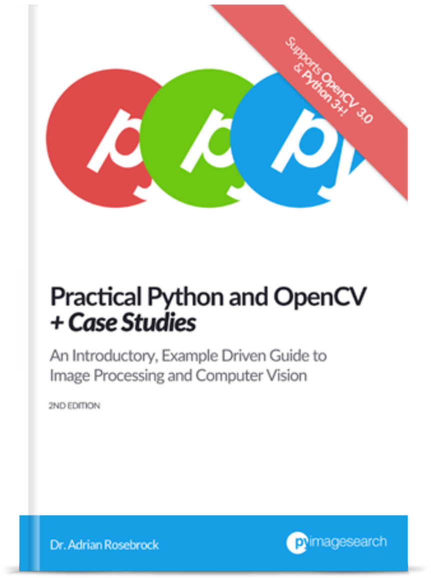
Click the button below to learn more about the course, take a tour, and get 10 (FREE) sample lessons.

TAKE A TOUR & GET 10 (FREE) LESSONS

Hello! I'm Adrian Rosebrock.

I'm an entrepreneur and Ph.D who has launched two successful image search engines, ID My Pill and Chic Engine. I'm here to share my tips, tricks, and hacks I've learned along the way.

Learn computer vision in a single weekend.


Supports OpenCV 3.0 & Python 3+!
Practical Python and OpenCV
+ Case Studies
An Introductory, Example Driven Guide to Image Processing and Computer Vision
2ND EDITION
Dr. Adrian Rosebrock
pyimagesearch

Want to learn computer vision & OpenCV? I can teach you in a **single weekend**. I know. It sounds crazy, but it’s no joke. My new book is your **guaranteed, quick-start guide** to becoming an OpenCV Ninja. So why not give it a try? Click here to become a computer vision ninja.

CLICK HERE TO BECOME AN OPENCV NINJA

Subscribe via RSS

Never miss a post! Subscribe to the PyImageSearch RSS Feed and keep up to date with my image search engine tutorials, tips, and tricks

POPULAR

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3
APRIL 18, 2016

Raspbian Stretch: Install OpenCV 3 + Python on your Raspberry Pi
SEPTEMBER 4, 2017

Install OpenCV and Python on your Raspberry Pi 2 and B+
FEBRUARY 23, 2015

Home surveillance and motion detection with the Raspberry Pi, Python, OpenCV, and Dropbox
JUNE 1, 2015

Ubuntu 16.04: How to install OpenCV
OCTOBER 24, 2016

How to install OpenCV 3 on Raspbian Jessie
OCTOBER 26, 2015

Basic motion detection and tracking with Python and OpenCV
MAY 25, 2015

Find me on **Twitter**, **Facebook**, **Google+**, and **LinkedIn**.