Wednesday, January 13, 2016

How to create a beautiful pencil sketch effect with OpenCV and Python

Interesting image filter effects, such as a pencil sketch or a cartoonizer effect, do not have to be very computationally involved to look good. In fact, in order to create a beautiful black-and-white pencil sketch effect, all you essentially need is some blurring and two image blending techniques called dodging and burning.

Using OpenCV and Python, an RGB color image can be converted into a pencil sketch in four simple steps:

1. Convert the RGB color image to grayscale.
2. Invert the grayscale image to get a negative.
3. Apply a Gaussian blur to the negative from step 2.
4. Blend the grayscale image from step 1 with the blurred negative from step 3 using a color dodge.

Whereas the first three steps are straightforward, I have seen other bloggers struggling with the fourth step, because OpenCV does not offer a native function to implement dodging and burning. But, with a little insight and a few tricks, we will arrive at our own implementation that in the end will look deceptively simple.

Step 1: Convert the color image to grayscale

This should be really easy to do even for an OpenCV novice. Images can be opened with `cv2.imread` and can be converted between color spaces with `cv2.cvtColor`. Alternatively, you can pass an additional argument to `cv2.imread` that specifies the color mode in which to open the image.

```
import cv2

img_rgb = cv2.imread("img_example.jpg")
img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2GRAY)
```

Step 2: Obtain a negative

A negative of the image can be obtained by "inverting" the grayscale value of every pixel. Since by default grayscale values are represented as integers in the range [0,255] (i.e., precision `CV_8U`), the "inverse" of a grayscale value `x` is simply `255-x`:

```
img_gray_inv = 255 - img_gray
```

Step 3: Apply a Gaussian blur

A Gaussian blur is an effective way to both reduce noise and reduce the amount of detail in an image (also called smoothing an image). Mathematically it is equivalent to convolving

Michael Beyeler

I am a Joint Moore/Sloan/WRF Innovation in Neuroengineering and Data Science Postdoctoral Fellow in the eScience Institute and the Institute for Neuroengineering (UWIN) at the University of Washington, where I am working on computational models of bionic vision, with the goal of improving the perceptual experience of blind patients implanted with a retinal prosthesis ("bionic eye").

Most Popular This Month

1. How to connect a non-Unifying Logitech keyboard & mouse to a new dongle
2. How to install Ubuntu 16.04 alongside Windows 10 (dual boot)
3. How to install CUDA 9 and CuDNN 7 on Ubuntu 18.04
4. How to create a beautiful pencil sketch effect with OpenCV and Python

an image with a Gaussian kernel. The size of the Gaussian kernel can be passed to `cv2.GaussianBlur` as an optional argument `ksize`. If both `sigmaX` and `sigmaY` are set to zero, the width of the Gaussian kernel will be derived from `ksize`:

```
img_blur = cv2.GaussianBlur(img_gray_inv, ksize=(21, 21),
                            sigmaX=0, sigmaY=0)
```

Step 4: Blend the grayscale image with the blurred negative

This is where things can get a little tricky. Dodging and burning refer to techniques employed during the printing process in traditional photography. In the good old days of traditional photography, people would try to lighten or darken a certain area of a darkroom print by manipulating its exposure time. Dodging lightened an image, whereas burning darkened it.

Modern image editing tools such as Photoshop offer ways to mimic these traditional techniques. For example, color dodging of an image `A` with a mask `B` is implemented as follows:

```
((B[idx] == 255) ? B[idx] : min(255, ((A[idx] << 8) / (255-B[idx]))))
```

This is essentially dividing the grayscale (or channel) value of an image pixel `A[idx]` by the inverse of the mask pixel value `B[idx]`, while making sure that the resulting pixel value will be in the range [0,255] and that we do not divide by zero. We could translate this into a naïve Python function that accepts two OpenCV matrices (an `image` and a `mask`) and returns the `blend`ed mage:

```
import cv2
import numpy as np

def dodgeNaive(image, mask):
  # determine the shape of the input image
  width,height = image.shape [:2]

  # prepare output argument with same size as image
  blend = np.zeros((width,height), np.uint8)

  for col in xrange(width):
    for row in xrange(height):
      # do for every pixel
      if mask[c,r] == 255:
        # avoid division by zero
        blend[c,r] = 255
      else:
        # shift image pixel value by 8 bits
        # divide by the inverse of the mask
        tmp = (image[c,r] << 8) / (255-mask)

        # make sure resulting value stays within bounds
        if tmp > 255:
          tmp = 255
          blend[c,r] = tmp

  return blend
```

As you might have already guessed, although this code might be functionally correct, it will undoubtedly be horrendously slow. First, the function uses for-loops, which are almost always a bad idea in Python. Second, `numpy` arrays (the underlying format of OpenCV images in Python) are optimized for array calculations, so accessing and modifying each pixel `image[c,r]` separately will be really slow.

Instead, we should realize that the operation `<<8` is the same as multiplying the pixel value with the number 2^8=256, and that pixel-wise division can be achieved with `cv2.divide`. An improved version of the dodging function could thus look like this:

```
def dodgeV2(image, mask):
  return cv2.divide(image, 255-mask, scale=256)
```

We have reduced the dodge function to a single line! The function `dodgeV2` produces the same result as `dodgeNaive` but is orders of magnitude faster. In addition, `cv2.divide` automatically takes care of the division by zero, making the result 0 where `255-mask` is zero. A burning function can be implemented analogously:

```
def burnV2(image, mask):
  return 255 – cv2.divide(255-image, 255-mask, scale=256)
```

With these tricks in our bag, we can now complete the pencil sketch transformation:

```
img_blend = dodgeV2(img_gray, img_blur)
cv2.imshow("pencil sketch", img_blend)
```

For kicks and giggles, we want to lightly blend the transformed image (`img_blend`) with a background image (`img_canvas`) that makes it look like we drew the image on a canvas:

```
img_canvas = cv2.imread("img_canvas.jpg")
img_blend = cv2.multiply(img_blend, img_canvas, scale=1/256)
```

The result looks like this:

5 How to de-noise images in Python

6 12 advanced Git commands I wish my co-workers would know

7 How to create a cool cartoon effect with OpenCV and Python

8 How to classify iris species using logistic regression

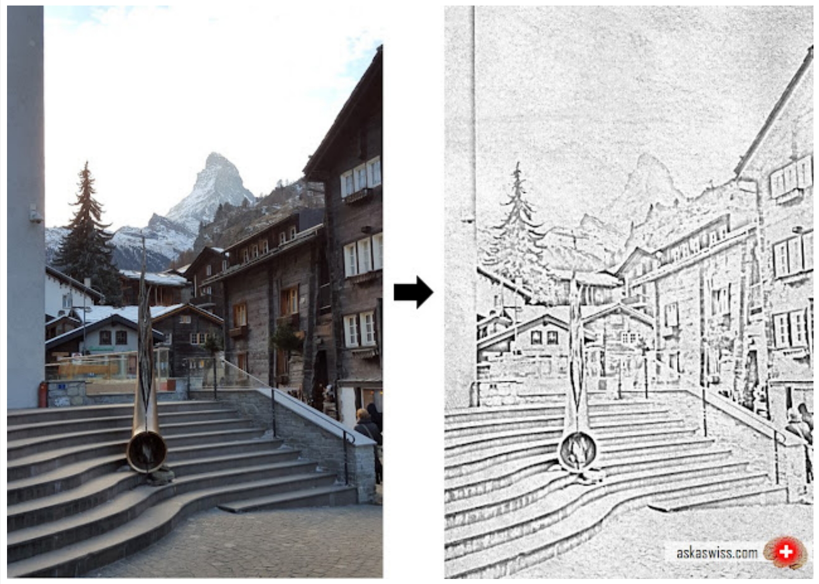
9 Open science gone awry: How 70,000 OKCupid users just had their private data exposed

Labels

AI Alzheimer's anomaly detection anxiety artificial vision BCI beginners big data books brain theories BrainScaleS classification clustering computational neuroscience computer vision computers conferences consciousness CUDA CuDNN data mining data science deep learning drones git GitHub GPU howto image processing installation ipython java linux machine learning matlab mental disorders mind reading motion neuroengineering neuroethics neuromorphic computing neurorobotics neuroscience nohup NVIDIA open access open science open source OpenCV Parkinson's pattern recognition programming Python Python Anaconda R regression reinforcement learning remote sale science scikit-image scikit-learn screen shell software SpiNNaker ssh statistics supervised learning top10 TrueNorth ubuntu unix unsupervised learning vision windows

Archive

- ► 2020 (1)
- ► 2019 (2)
- ► 2018 (8)
- ► 2017 (20)
- ▼ 2016 (43)
  - ► December (4)
  - ► November (4)
  - ► October (4)
  - ► September (2)
  - ► August (4)
  - ► June (3)
  - ► May (4)
  - ► April (2)
  - ► March (6)
  - ► February (2)
  - ▼ January (8)
    - Highlights and new discoveries in Computer Vision,...
    - Beginners' 10 most common Java programming mistake...
    - How to install OpenCV 3.1 for Python on Ubuntu 14.04
    - How to create a beautiful pencil sketch effect wit...
    - 7 tips & tricks to get you started on the Unix com...



Did you notice that the procedure can be further optimized? You'll find the answer in the book OpenCV with Python Blueprints. ;-) All source code is available for free on GitHub (refer to the `PencilSketch` class in the `filters` module).

Posted by Unknown at 7:40 AM

Follow @mbeyelerCH

computer vision , howto , image processing , OpenCV , programming , Python

No comments :

Post a Comment

Enter your comment...

Comment as: karthik8297@g

Sign out

Publish Preview

Notify me