Has anyone converted Matlab / C++ gradient code into Python?

little **Aug 18**

Just wondering if anyone here may provide a quick tip for converting this gradient calculation method into Python-OpenCV?

So far have been using the inbuilt Sobel, Scharr and Laplacian operators in OpenCV-Python and have not had any experience with custom gradients yet.

Thanks!

How it looks like in Matlab

> [x(2)-x(1) (x(3:end)-x(1:end-2))/2 x(end)-x(end-1)] with x being the input

How it looks like in C++

```
cv::Mat computeMatXGradient(const cv::Mat &mat) {
cv::Mat out(mat.rows,mat.cols,CV_64F);

for (int y = 0; y < mat.rows; ++y) {
const uchar *Mr = mat.ptr(y);
double *Or = out.ptr(y);

 Or[0] = Mr[1] - Mr[0];
 for (int x = 1; x < mat.cols - 1; ++x) {
   Or[x] = (Mr[x+1] - Mr[x-1])/2.0;
 }
 Or[mat.cols-1] = Mr[mat.cols-1] - Mr[mat.cols-2];

}

return out;
}
```

ThamNgapWei **Aug 18**

This problem is quite easy to solve, in the world of python, "numpy" is all you need for this kind of task.

Adrian Chief PyImageSearcher **Aug 18**

In Python, you would try to avoid any type of `for` loop that loops over the rows and columns of the image. This is mainly because Python loops and pixel lookups are painfully slow compared to their C/C++ counterparts.

Instead, you try to use NumPy and vectorized operations whenever you can.

Take a look at the HOG implementation of scikit-image were they use NumPy to quickly compute the gradient:

scikit-image/scikit-image/blob/master/skimage/feature/_hog.py#L105

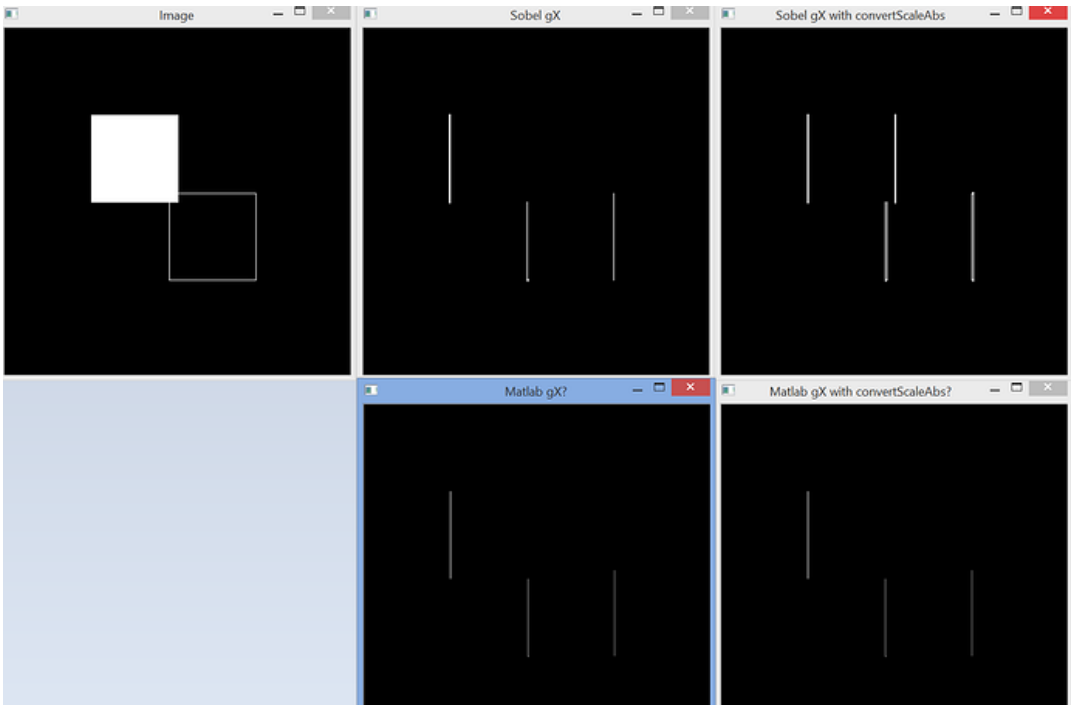
```
95  extent. Variant methods may also include second order image derivative
96  which act as primitive bar detectors - a useful feature for capturing,
97  e.g. bar like structures in bicycles and limbs in humans.
98  """
99
100 if image.dtype.kind == 'u':
101     # convert uint image to float
102     # to avoid problems with subtracting unsigned numbers in np.diff()
103     image = image.astype('float')
104
105 gx = np.empty(image.shape, dtype=np.double)
106 gx[:, 0] = 0
107 gx[:, -1] = 0
108 gx[:, 1:-1] = image[:, 2:] - image[:, :-2]
109 gy = np.empty(image.shape, dtype=np.double)
110 gy[0, :] = 0
111 gy[-1, :] = 0
112 gy[1:-1, :] = image[2:, :] - image[:-2, :]
113
114 """
115 The third stage aims to produce an encoding that is sensitive to
```

---

little **Aug 20**

Thanks @Adrian for the excellent example!
Thanks @ThamNgapWei also.

The example helped to open my mind which was kind of rusty a few days back.
Below are the findings (finer/thinner "gradient lines" vs. the default Sobel), probably due to the division /2.0.
Also, it "highlights" black transiting to white, while it does not seem to "highlight" white transiting to black.
Side thought: would be nice to experiment more with gradients subsequently.



Sharing the code below in case anyone is looking out for something like that
(please feel free to correct in case I did the code conversion to python wrongly)

```
#takes in image that is already in numpy format

def gradiantMatlab(image):
    img = image.copy()
    img[:, 0]  = img[:, 1] - img[:, 0]
    img[:, 1:-1] = (img[:, 2:] - img[:, :-2])/2
    img[:, -1] = img[:, -1]- img[:, -2]
    return img
```

Adrian Chief PyImageSearcher **Aug 20**

What data type is your `image`? I presume an unsigned 8-bit integer? Convert it to a floating point type first:

```
image = image.astype("float")
```

And then apply your operation. To display the image via `cv2.imshow`, you'll need to convert it back to `uint8`.

Take a look at the "One Important Matter!" section of this tutorial:

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_gradients/py_gradients.html#gradients

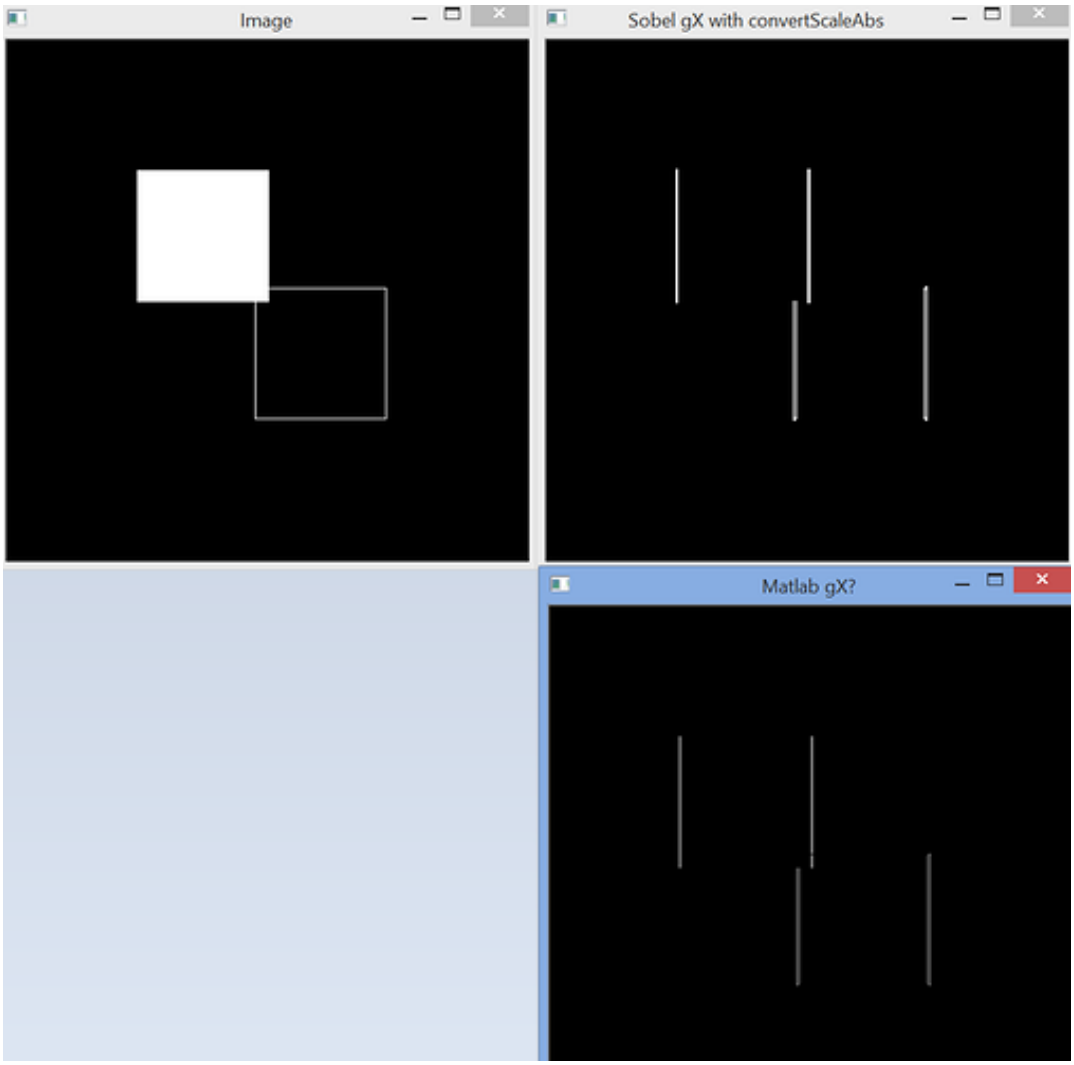
And you'll see why the negative slop won't be picked up by the unsigned 8-bit integer data type.

---

little **Aug 21**

Thanks a lot **@Adrian** !

Have fixed the bug. The result for the gradient looks pretty precise compared to the default Sobel, including the little gap near the bottom of the line at the center of the scene (because the white line extends from the extreme left to the extreme right in the original picture)



Updated code

```
def gradiantMatlab(image):
    img = image.copy()
    if image.dtype.kind == 'u':
        img = img.astype('float')
    img[:, 0]  = img[:, 1] - img[:, 0]
    img[:, 1:-1] = (img[:, 2:] - img[:, :-2])/2.0
    img[:, -1] = img[:, -1]- img[:, -2]
    img = img.astype('uint8')
    return img
```

Adrian Chief PyImageSearcher **Aug 22**

No problem, I'm happy I could help out there. That floating point "bug" is a really, really hard one to catch.