浙江大学爱丁堡大学联合学院

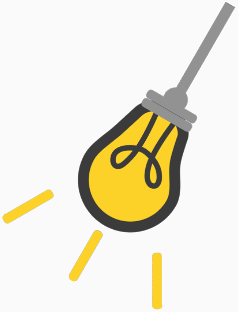
ZJU-UoE Institute

Lecture 5 - Edge detection

Nicola Romanò - nicola.romano@ed.ac.uk

Learning objectives

- Explain the use of image derivatives for edge detection.
- Describe various edge detection algorithms and how they work.
- Use Scikit Image to implement these techniques.



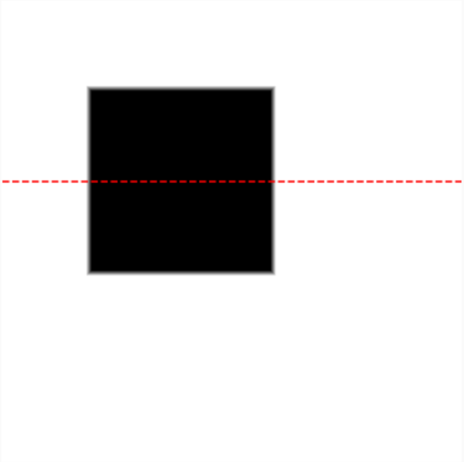
Edge detection problem

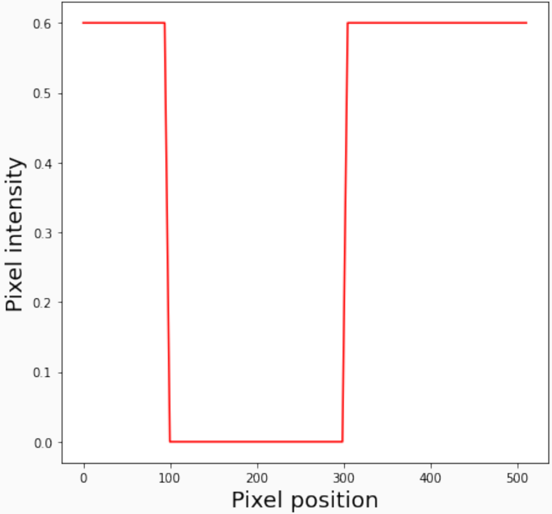
An edge is an area where the brightness of an image changes more or less gradually.

Detecting edges is useful e.g. to find objects in a scene, determine which pixels belong to which objects, and measure their properties.

Monarch butterfly - CC-BY-SA 2.0 Ted @ Flickr

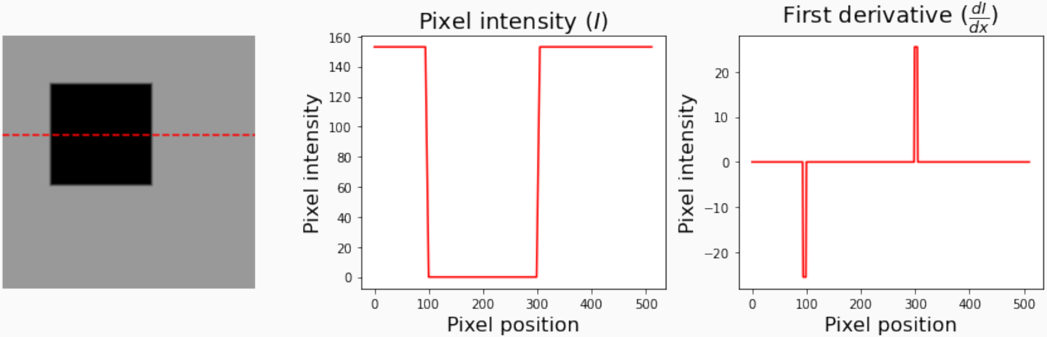
How to detect an edge?





Consider the vertical edges. We can see a change in the intensity of pixels as we move from black to white and vice versa. **Can you think of a way to detect these edges?**

We can use derivatives!



Edges will correspond to minima and maxima of the derivative of the intensity!

Image derivatives

With a 2D image, we can find the x and y derivatives of the image intensity.

The vector of derivatives is called the **gradient** and is given by:

$$\nabla I = \left[\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right]$$

The gradient is a vector with:

- Direction perpendicular to the edge

$$\theta = \arctan\left(\frac{\partial I}{\partial x} \Big/ \frac{\partial I}{\partial y}\right)$$

Image derivatives

With a 2D image, we can find the x and y derivatives of the image intensity.

The vector of derivatives is called the **gradient** and is given by:

$$\nabla I = \left[\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right]$$

The gradient is a vector with:

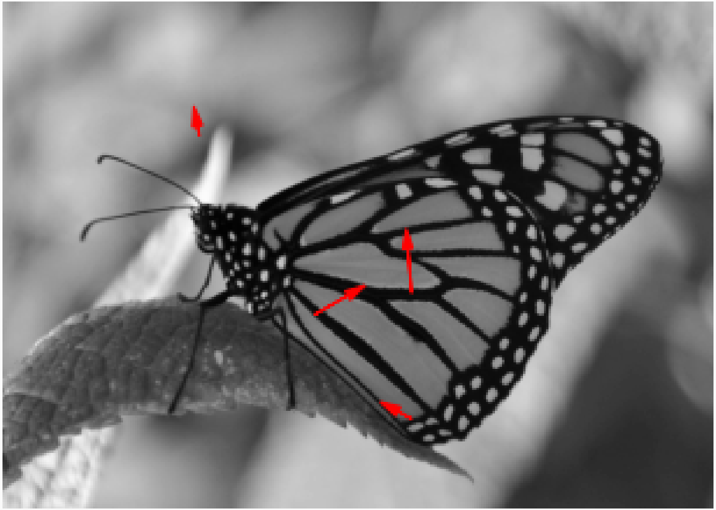
- Direction perpendicular to the edge

$$\theta = \arctan\left(\frac{\partial I}{\partial x} \bigg/ \frac{\partial I}{\partial y}\right)$$

- Length (**gradient magnitude**) proportional to the intensity change

$$||\nabla I|| = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2}$$

Example of gradient vectors



How to calculate a discrete derivative?

Remember the definition of a derivative for a continuous function $f(x)$:

$$f'(x) = \frac{df}{dx} = \lim_{\Delta x \to 0} \frac{f(x) - f(x - \Delta x)}{\Delta x}$$

Calculating a discrete derivative

How to calculate a discrete derivative?

Remember the definition of a derivative for a continuous function $f(x)$:

$$f'(x) = \frac{df}{dx} = \lim_{\Delta x \to 0} \frac{f(x) - f(x - \Delta x)}{\Delta x}$$

Our image is not continuous, as it is made up of discrete pixels so the minimum Δx value is 1 (a single pixel).

Calculating a discrete derivative

How to calculate a discrete derivative?

Remember the definition of a derivative for a continuous function $f(x)$:

$$f'(x) = \frac{df}{dx} = \lim_{\Delta x \to 0} \frac{f(x) - f(x - \Delta x)}{\Delta x}$$

Our image is not continuous, as it is made up of discrete pixels so the minimum Δx value is 1 (a single pixel).

The discrete derivative is given by:

$$\frac{dI}{dx} = \frac{I(x) - I(x-1)}{1} = I(x) - I(x-1)$$

We can choose to calculate the discrete derivative in three different ways

- **Forward difference**: $I(x) - I(x+1)$
- **Backward difference**: $I(x) - I(x-1)$
- **Central difference**: $I(x+1) - I(x-1)$

We can choose to calculate the discrete derivative in three different ways

- **Forward difference**: $I(x) - I(x+1)$
- **Backward difference**: $I(x) - I(x-1)$
- **Central difference**: $I(x+1) - I(x-1)$

These can be easily calculated using **convolution**!

- **Forward difference**: $\begin{bmatrix} 1 & -1 \end{bmatrix}$
- **Backward difference**: $\begin{bmatrix} -1 & 1 \end{bmatrix}$
- **Central difference**: $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$

Discrete derivative - example

Let's calculate the discrete derivative of this 1D image using central difference

10	16	22	36	40	11	17	23	37	41

Let's calculate the discrete derivative of this 1D image using central difference

10	16	22	36	40	11	17	23	37	41

Convolving with the kernel

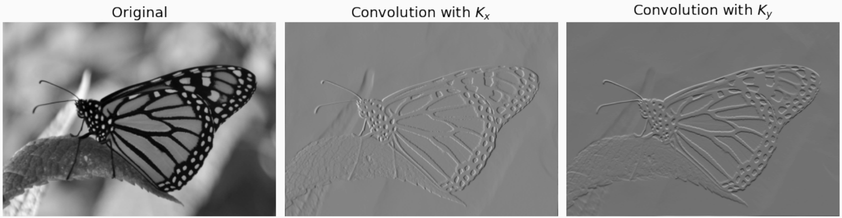
-1	0	1

We obtain the following

0	12	20	18	-25	-23	12	20	18	0

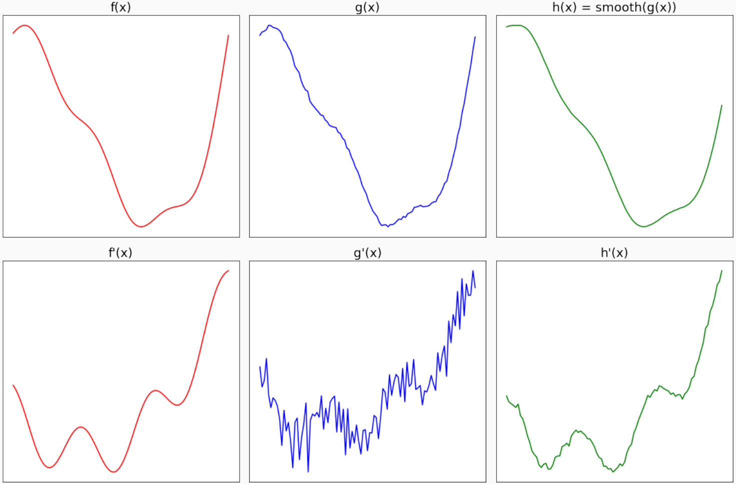
We can apply these convolution kernels to an image as well.

$$K_x = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \qquad K_y = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$



The problem with noise...

Derivatives are very sensitive to noise. Smoothing the function beforehand helps



Outline

2D derivative kernels (Prewitt edge detectors)

Expanding the derivative kernels to 2D we can directly smooth our intensity function.

These are called **Prewitt** kernels.

$$K_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad K_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Alternatively the **Sobel** kernels can be used:

$$K_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad K_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

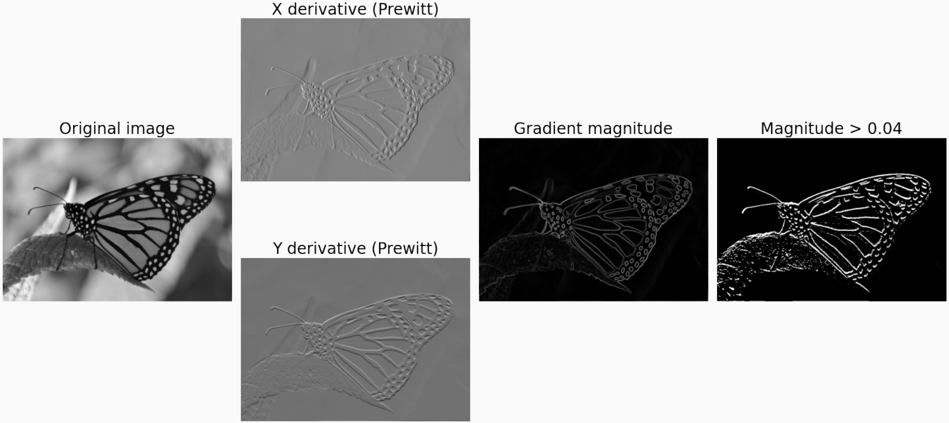
Your turn!

Consider the following image. **What do you expect to obtain and why** after convolution with the Prewitt kernels?
Apply the convolution (you can easily do that by hand); **do the results match** your prediction?

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 50 & 50 & 50 & 50 & 50 & 50 & 50 & 50 \\ 100 & 100 & 100 & 100 & 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 & 100 & 100 & 100 & 100 \\ 50 & 50 & 50 & 50 & 50 & 50 & 50 & 50 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Having applied either the Prewitt or Sobel kernels to the image we can now detect edges.

Simply threshold the gradient magnitude of the image to define which pixels are edges.

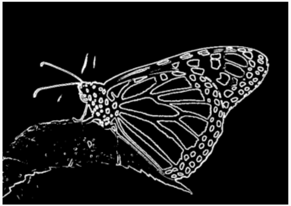


Edge detection in Scikit Image

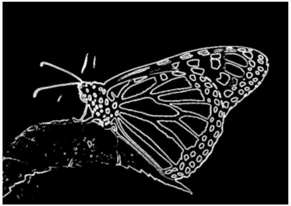
```
from skimage.filters import prewitt, sobel
from skimage.io import imread

img = imread("butterfly.jpg")
im_prewitt = prewitt(img)
im_sobel = sobel(img)
```

Prewitt



Sobel

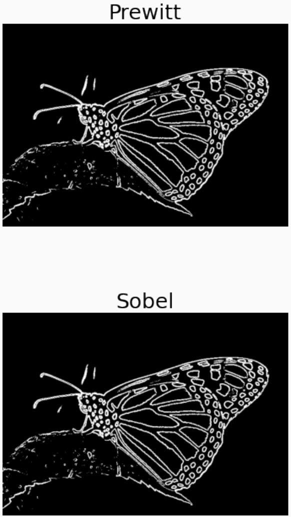


Edge detection in Scikit Image

```
from skimage.filters import prewitt, sobel
from skimage.io import imread

img = imread("butterfly.jpg")
im_prewitt = prewitt(img)
im_sobel = sobel(img)

fig, ax = plt.subplots(2, 1, figsize=(5, 10))
ax[0].imshow(im_prewitt > 0.08, cmap="gray")
ax[0].set_title("Prewitt", fontsize=25)
ax[1].imshow(im_sobel > 0.08, cmap="gray")
ax[1].set_title("Sobel", fontsize=25)
for a in ax:
a.axis("off")
plt.show()
```



The Canny edge detector

The Canny edge detector is a more advanced algorithm to detect edges.

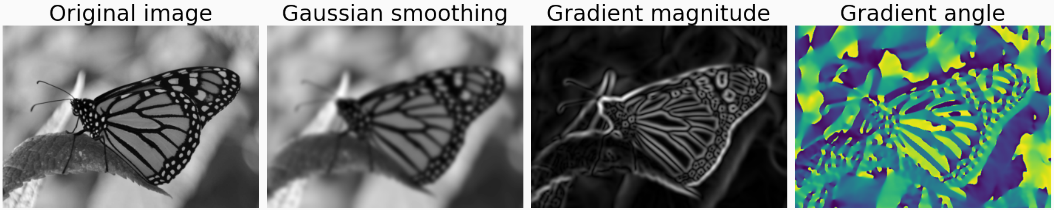
It involves five steps

1. Apply a Gaussian filter to the image to smooth out noise
2. Calculate the gradient magnitude (e.g. using a Sobel filter)
3. Non-maximum suppression
4. Double thresholding
5. Edge tracking by hysteresis

The Canny edge detector - step 1 and 2 - smoothing and gradient

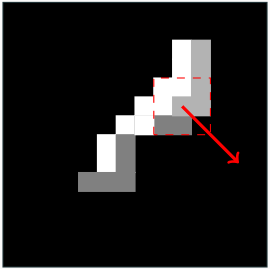
We start by convolving the image with a Gaussian kernel to smooth noise.

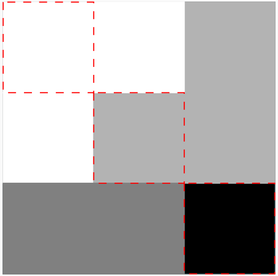
We then calculate the gradient magnitude and angle using the Sobel kernels.



Non-maximum suppression allows us to thin the edges by only keeping the pixels with the largest gradient magnitude in the edge.

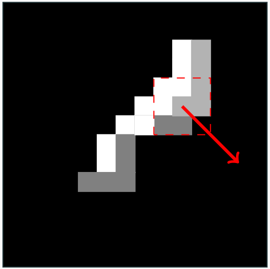
For each pixel, we take the neighbouring pixels in the direction of the gradients, and we keep only the pixels with the largest gradient magnitude.

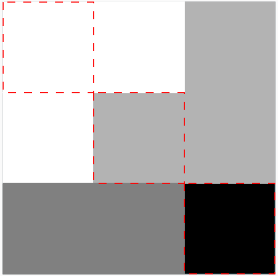


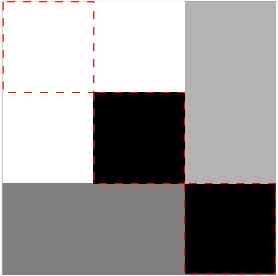


Non-maximum suppression allows us to thin the edges by only keeping the pixels with the largest gradient magnitude in the edge.

For each pixel, we take the neighbouring pixels in the direction of the gradients, and we keep only the pixels with the largest gradient magnitude.







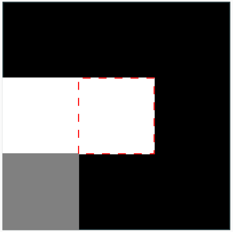
The Canny edge detector - step 4 - double thresholding

The next step is to set two arbitrary thresholds, one for the weak edges and one for the strong edges.

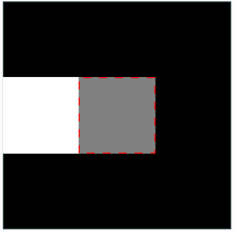
- **Strong edges** are those with gradient magnitude above the high threshold.
- **Weak edges** are those with gradient magnitude between the low and high threshold.
- Edges with gradient magnitude below the low threshold are suppressed (set to 0).

The Canny edge detector - step 5 - hysteresis

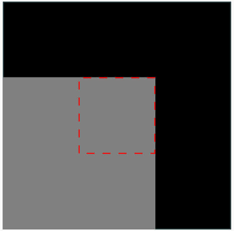
We need to decide what to do with weak edges. We keep those weak edges that are near a strong edge, and discard the others.



Strong
edge, keep

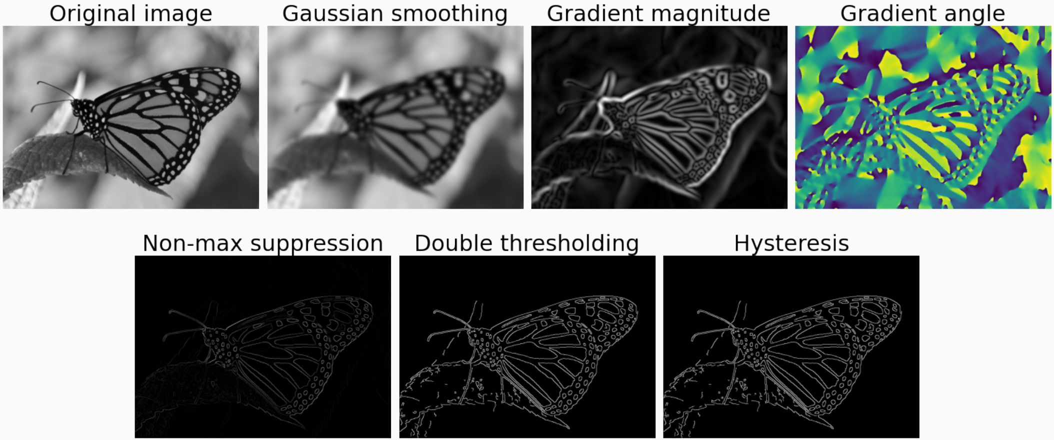


Weak edge near
strong, mark as strong



Weak edge,
remove

The Canny edge detector - The final result!



This is implemented in the skimage.feature.canny function. Try it by yourself and change the parameters to see what happens!

Want to read more? The original 1986 paper from Canny is attached (lots of maths in there!).

Summary

- Edge detection is a valuable tool for image processing.
- We have covered some of the most common algorithms for edge detection, which can be used as an early step in more complex analysis pipelines.
- In the next lecture we will look at other algorithms to detect specific features in images.