Les données numériques: corrélation pour les images

Diane Lingrand

Polytech SI3

2022 - 2023

Outline

Outline

Cross-correlation

- For f and g discrete functions :

$$f \star g(x,y) = \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} f(x,y) g(x-u, y-v)$$

- Applied to image I_1 by kernel k of shape $(2p+1)x(2q+1)$:

$$I_2[x,y] = I_1 \star k[x,y]$$

$$= \sum_{i=0}^{2p} \sum_{j=0}^{2q} I_1[x+i-p, y+j-q] k[i][j]$$

  - same kernel applied to different locations on the image
  - equivalent to convolution if the kernel is symetric (horizontally and vertically).

Toy example

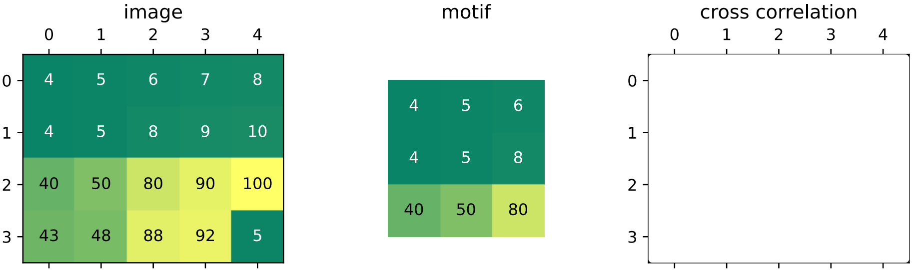


image
0 1 2 3 4
4 5 6 7 8
4 5 8 9 10
40 50 80 90 100
43 48 88 92 5
motif
4 5 6
4 5 8
40 50 80
cross correlation
0 1 2 3 4

Toy example

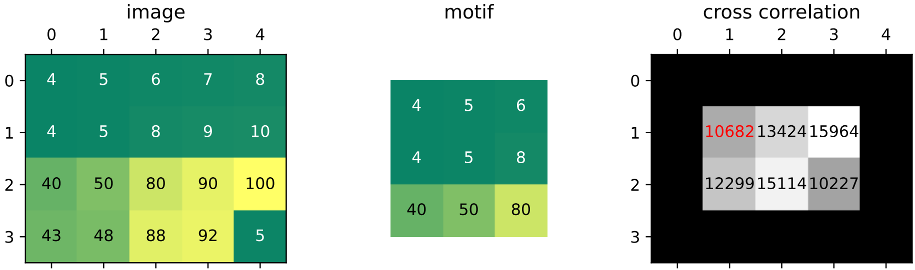


image
0 1 2 3 4
4 5 6 7 8
4 5 8 9 10
40 50 80 90 100
43 48 88 92 5
motif
4 5 6
4 5 8
40 50 80
cross correlation
0 1 2 3 4
10682 13424 15964
12299 15114 10227

Normalised Cross-correlation

- Cross-Correlation of image I_1 by kernel k of shape $(2p+1)x(2q+1)$ $(N = (2p+1)(2q+1))$:

$$I_2[x,y] = I_1 \star k[x,y]$$

$$= \frac{1}{\sigma_k \sigma_{Ixy} N} \sum_{i=0}^{2p} \sum_{j=0}^{2q} (I_1[x+i-p, y+j-q] - \mu_{Ixy})(K[i][j] - \mu_k)$$

  - where :
    - μ_k is the mean of kernel values
    - σ_k is the standard deviation of kernel values
    - μ_{Ixy} (resp. σ_{Ixy}) is the mean (resp. standard deviation) of pixel values in the neighborhoood of coordinates $(x;y)$. The dimensions of the neighborhood are those of the kernel.
  - values between -1 and +1.

Toy example

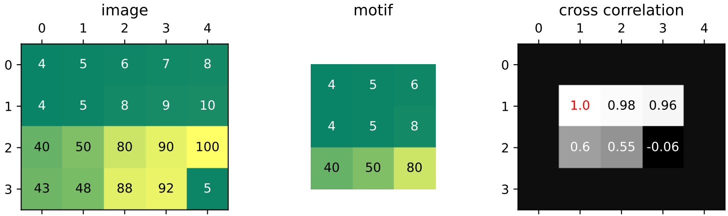
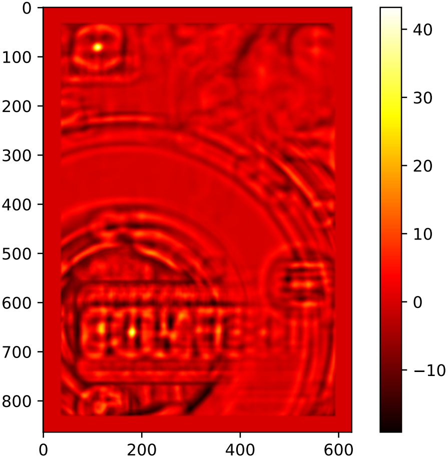


image
0 1 2 3 4
0 4 5 6 7 8
1 4 5 8 9 10
2 40 50 80 90 100
3 43 48 88 92 5
motif
4 5 6
4 5 8
40 50 80
cross correlation
0 1 2 3 4
0
1 1.0 0.98 0.96
2 0.6 0.55 -0.06
3

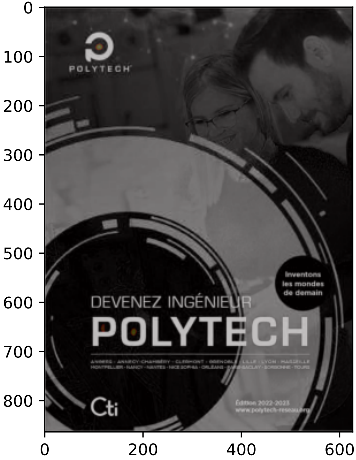
Find a logo in an image

Find in

Result



Grey levels. Correlation results plotted using hot colormap. Max 0.999 at (82 ;110).



Superposition of grey image with highest values from the correlation map (> 0.5)

Naive implementation

```
def crossCorrelation(im, mo):
    hmo, wmo = mo.shape[:2]
    hmo2, wmo2 = hmo//2, wmo//2 # assuming mo of odd width and height
    him, wim = im.shape[:2]
    res = np.zeros(im.shape)
    mo = mo-np.mean(mo)
    stdMo = math.sqrt(np.mean(mo*mo))
    if stdMo == 0:
        return res
    mo /= stdMo
    for j in range(wmo2, wim-wmo2):
        for i in range(hmo2, him-hmo2):
            #mean of roi
            meanRoi = 0
            for l in range(wmo):
                for k in range(hmo):
                    meanRoi += im[i+k-hmo2][j+l-wmo2]
            meanRoi /= hmo*wmo
            # standard dev of roi
            stdRoi = 0
            for l in range(wmo):
                for k in range(hmo):
                    stdRoi += (im[i+k-hmo2][j+l-wmo2]-meanRoi)**2
            stdRoi = math.sqrt(stdRoi/hmo/wmo)
            if stdRoi == 0:
                res[i][j] = 0
            #cross correlation
            else:
                for l in range(wmo):
                    for k in range(hmo):
                        res[i][j] += (im[i+k-hmo2][j+l-wmo2]-meanRoi)*mo[k][l]
                res[i][j] /= stdRoi
    res /= hmo*wmo
    return res
```

Improve the implementation

- numpy functions are faster
  - compute mean and standard deviation using numpy
  - compute correlation as a pointwise multiplication in numpy
- test the previous example
  - and observe the difference

```python
print(a, "\t\tmatrix a\n")
print(b, "\t matrix b\n")
print(a*b, "\t pointwise multiplication\n")
print(a@b, "\t matrix multiplication")
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]                matrix a

[[  10   20   30]
 [ 200  100  200]
 [1000 2000 3000]]       matrix b

[[   10    40    90]
 [  800   500  1200]
 [ 7000 16000 27000]]    pointwise multiplication

[[ 3410  6220  9430]
 [ 7040 12580 19120]
 [10670 18940 28810]]    matrix multiplication
```

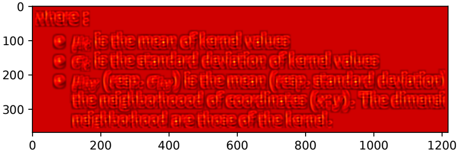
Find letters in a text

Find a's in

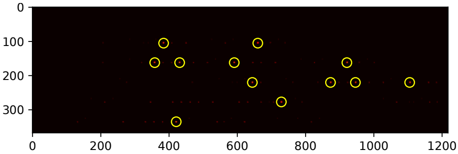
where :

- μ_k is the mean of kernel values
- σ_k is the standard deviation of kernel values
- μ_{Ixy} (resp. σ_{Ixy}) is the mean (resp. standard deviation) the neighborhoood of coordinates $(x; y)$. The dimensic neighborhood are those of the kernel.

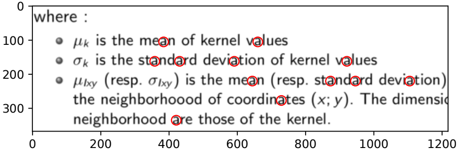
correlation map



correlation values > 0.5
highlighted with yellow circles



localisation of max. correlations
on the original image



Your job today

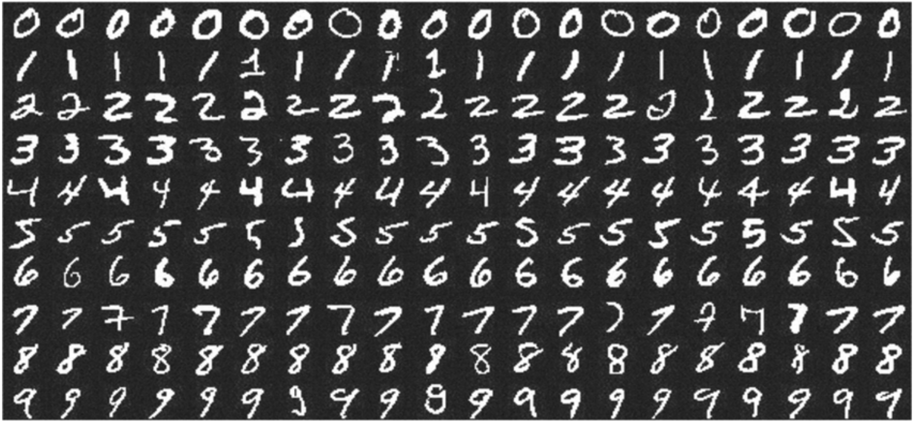
- Implement the correlation without and with normalisation
  - do not forget to use `numpy` for improvements
- Choose an image with different instances of logos. It can be a text with letters.
  - compute and visualise the correlation map
  - find the highest values (`numpy.argwhere(condition)` can help)
  - redraw the highest values locations on the original image
    - for example, this was used on previous slide :

```
for (a,b) in lesA:
    c = Circle((b,a), color='yellow',radius=14, linewidth=1, fill=False)
    ax.add_patch(c)
```

Outline

MNIST dataset

Dataset of 60000+10000 grayscale squared images (28x28).



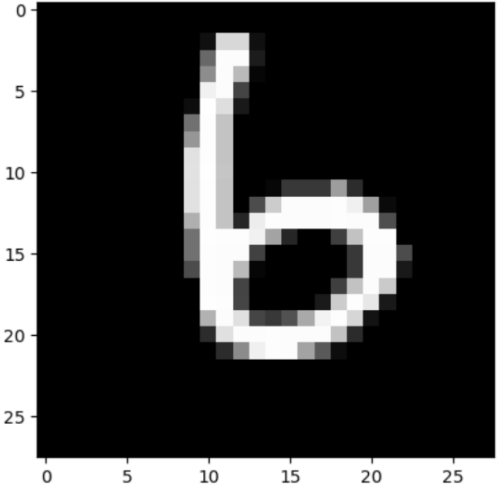
Reduced version of MNIST

- 10 classes of gray images (28x28)
- only 2000 images in the train set (200 per class)
- only 1000 images in the test set (100 per class)
- How to get the files ?
  - download :
    - https://www.i3s.unice.fr/~lingrand/redMNIST-x-train.bin
    - https://www.i3s.unice.fr/~lingrand/redMNIST-x-test.bin
    - https://www.i3s.unice.fr/~lingrand/redMNIST-y-train.bin
    - https://www.i3s.unice.fr/~lingrand/redMNIST-y-test.bin
  - load in you python code :

```
import pickle
with open('redMNIST-x-train.bin', 'rb') as input:
    xTrain = pickle.load(input)
with open('redMNIST-x-test.bin', 'rb') as input:
    xTest = pickle.load(input)
with open('redMNIST-y-test.bin', 'rb') as input:
    yTest = pickle.load(input)
with open('redMNIST-y-train.bin', 'rb') as input:
    yTrain = pickle.load(input)
```

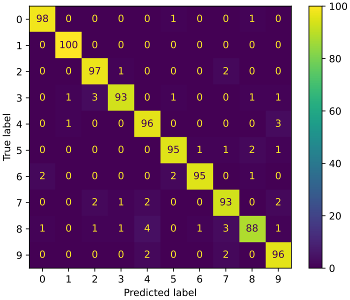
Looking at images

```
import random
n = random.randrange(0,len(yTrainr))
plt.imshow(xTrain[n], cmap=plt.cm.gray)
```



Final goal

- Confusion matrix computed on the test set :



- accuracy = 95.1%

How to do that ?

- main idea :
  - the results of correlation/convolution of some filters will help to represent an image in a more compact and mean full than the $28 * 28 = 784$ pixel values
- which filters ?
  - can be handcrafted : $\cap, \cup, \bigcirc, \subset, \supset, |, -, /$
  - I can give you some other filters
- It is not necessary to compute corr./conv. at every position :
  - skip some rows and cols
  - for example : 1 over 3 cols and 1 over 3 rows
    - it will speed the computations
- normalisation
  - divide every MNIST digit by 255
  - use almost normalised filters
  - and thus skip the normalisation of correlation in order to speed
  - eventually, remove all negative values
- when each image is represented using some corr./conv. results
  - perform a classification :
    - using kNN
    - using another algorithm that we will see next week : logistic regression

Why this approach ?

- It is a simplified version of a well known deep neural network for image classification :
  - CNN = Convolutional Neural Network
  - except that a CNN :
    - will learn the coefficient of corr./conv.
    - may stack different series of corr./conv. filters
    - may use a non linear classification
    - may use different other tricks
- You will learn true CNN next year
- Here are some filters you could test :