Lab 12

Basic Image Processing
Fall 2018

Warning!

On the Lecture you met with the HOG method
which is the topic of this lab practice.

The original HOG method is a publication of Navneet Dalal and Bill Triggs and it is mostly used for pedestrian recognition.

Today we are implementing a **SIMPLIFIED** version of the HOG!
Therefore you may notice differences between the article and this implementation.

Our implementation is referred as the
Poor Man's Histogram of Oriented Gradients
method.

Poor Man’s Histogram of Oriented Gradients

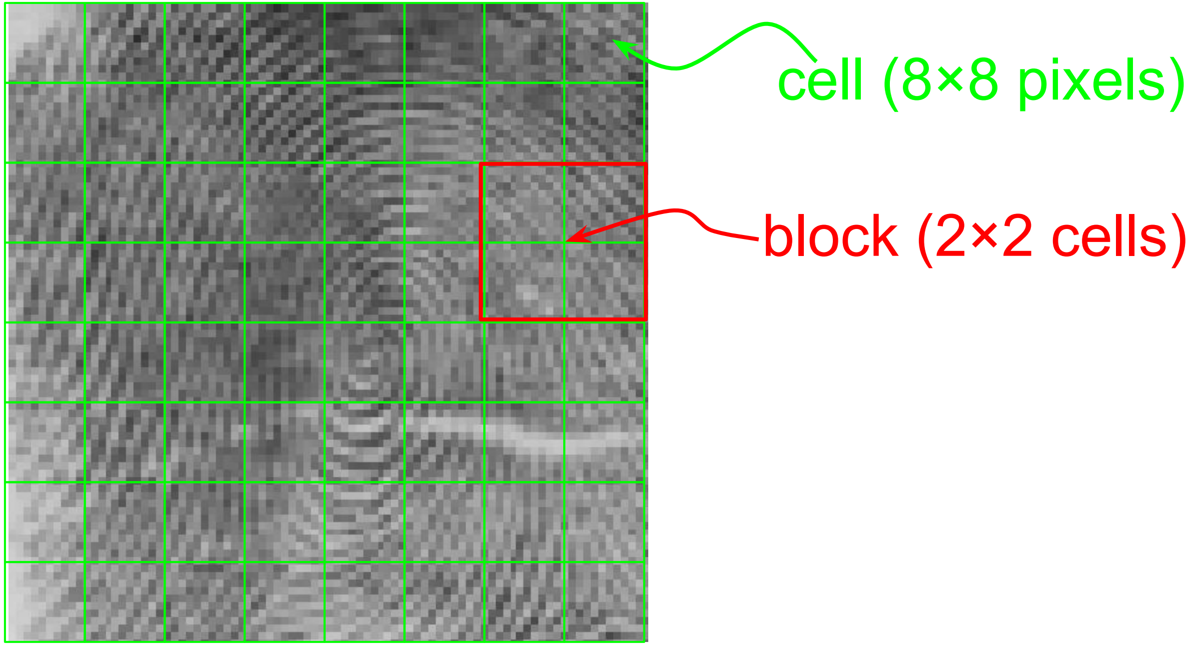
This algorithm is very similar to the original HOG method [1] but it contains some simplifications. The steps:

0. Creation of cells and blocks
1. Gradient computation
2. Orientation binning
3. Block normalization

On the next slides you find the description of the steps.

0. Creation of cells and blocks

The input image must be grayscale. We divide it into 8 × 8 pixel regions called cells and 2 cell × 2 cell regions called blocks:



1. Gradient computation

Gradient computation is carried out for each cell individually. It is done by applying an edge detection filter on the grayscale input image **G** both horizontally and vertically.

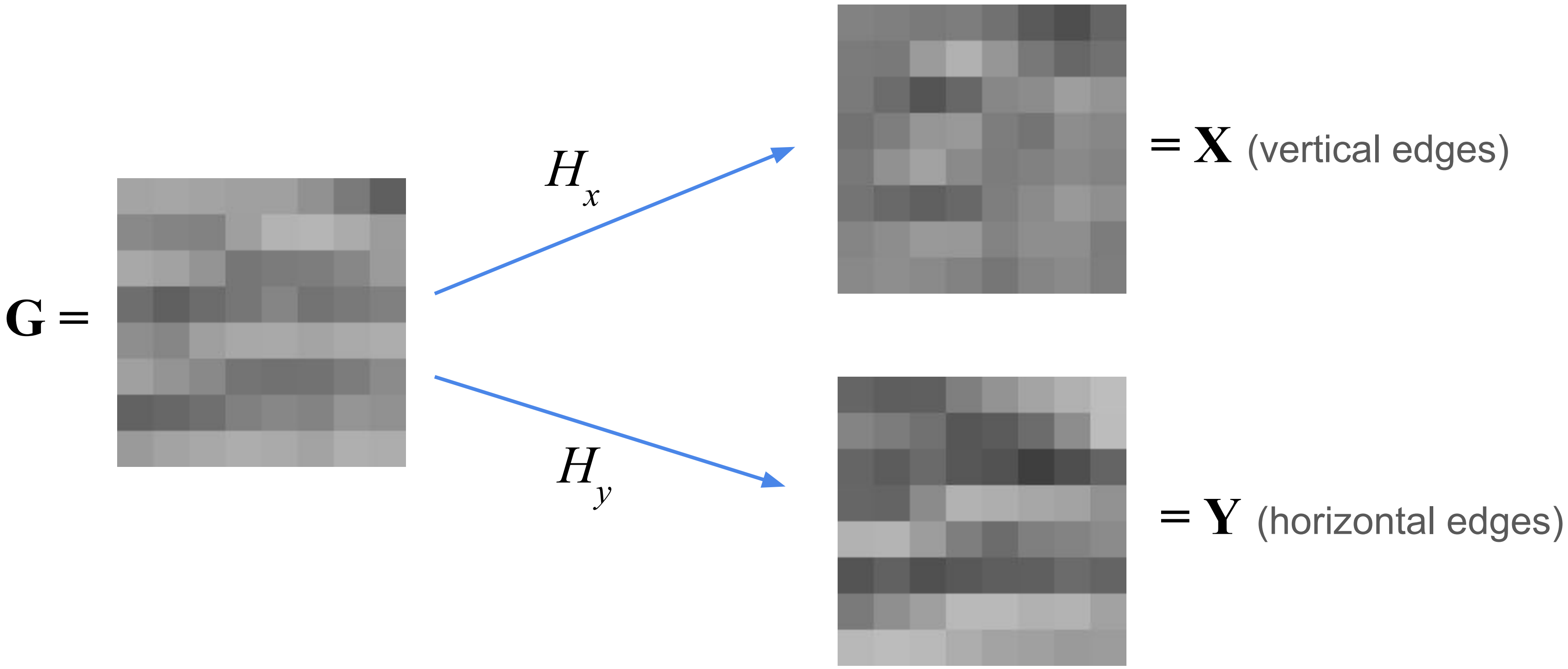
$$H_x = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \qquad \mathbf{X} = [x_{ij}] = \mathbf{G} \circledast H_x$$

$$H_y = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \qquad \mathbf{Y} = [y_{ij}] = \mathbf{G} \circledast H_y$$

Where **G, X, Y** are 8×8 size matrices (cells).

1. Gradient computation

illustration



$\mathbf{G} =$

H_x

H_y

$= \mathbf{X}$ (vertical edges)

$= \mathbf{Y}$ (horizontal edges)

2. Orientation binning part 1

The previous step resulted in two matrices $\mathbf{X}$ and $\mathbf{Y}$ in which the gradient vector's x and y component is stored for every (i, j) point.

So! At every (i, j) point we know two values: x_{ij} and y_{ij}. These are the vertical and horizontal components of the gradient vector. The vector's angle is

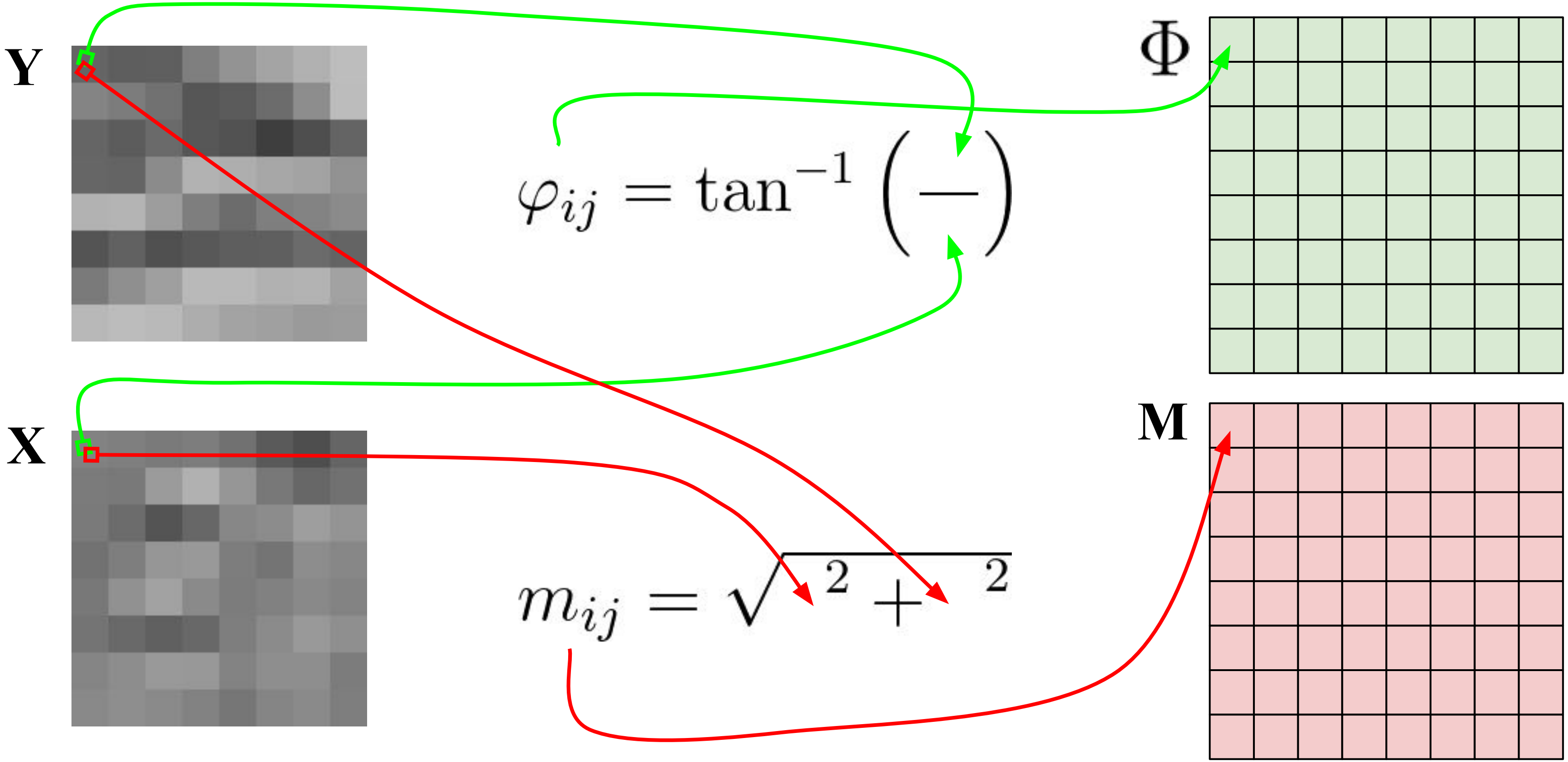
$$\varphi_{ij} = \tan^{-1}\left(\frac{y_{ij}}{x_{ij}}\right)$$

and the length of the gradient vector (magnitude) is

$$m_{ij} = \sqrt{x_{ij}^2 + y_{ij}^2}$$

2. Orientation binning

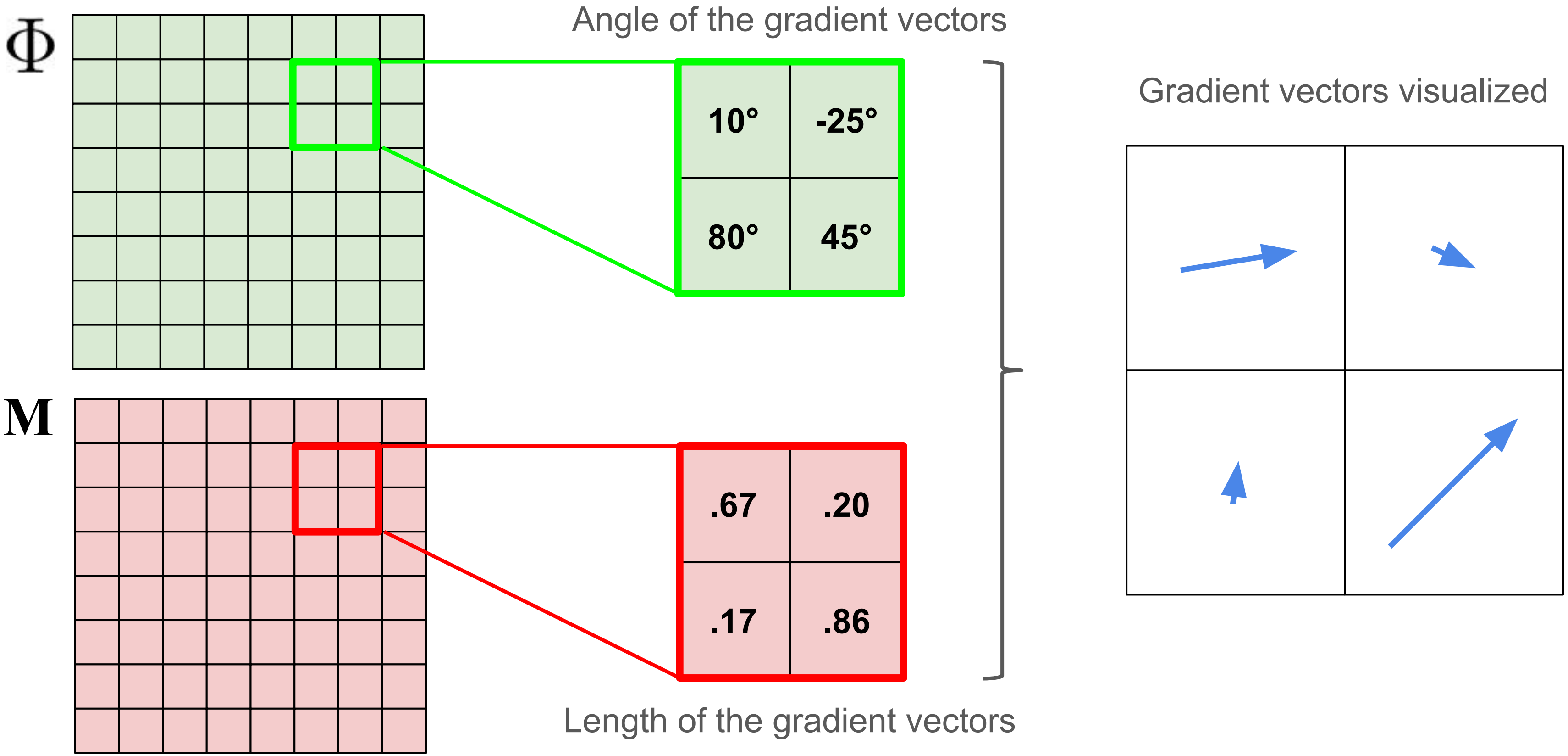
illustration part 1



$$\varphi_{ij} = \tan^{-1}\left(\frac{\quad}{\quad}\right)$$

$$m_{ij} = \sqrt{\quad^{2} + \quad^{2}}$$

2. Orientation binning illustration part 1



Angle of the gradient vectors

10°	-25°
80°	45°

Length of the gradient vectors

.67	.20
.17	.86

Gradient vectors visualized

2. Orientation binning part 2

After each angle and magnitude is calculated within the cell, a 9 bin histogram is initialized. The histogram’s bins define 9 intervals from -90° to 90°:

[-90, -70) [-70, -50) [-50, -30) [-30, -10) [-10, 10) [10, 30) [30, 50) [50, 70) [70, 90)

For every gradient vector in the cell we add the magnitude value to the appropriate bin.

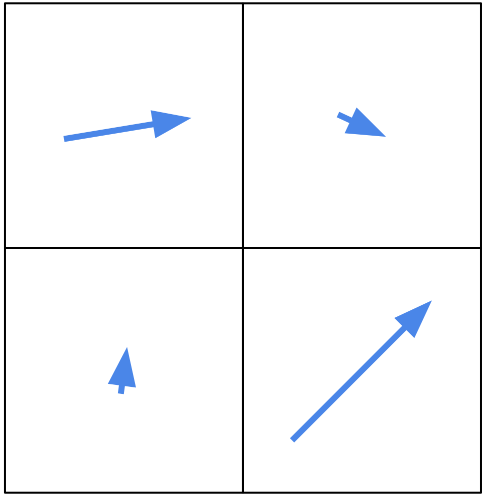
This procedure results in one orientation histogram per cell.

2. Orientation binning illustration part 2

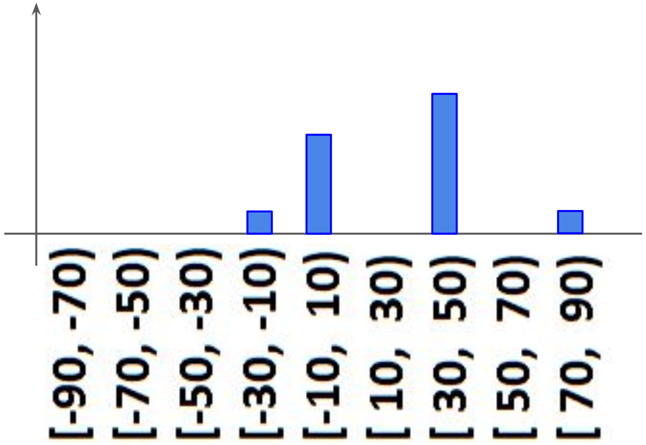
Angle of the gradient vectors

10°	-25°
80°	45°

Gradient vectors visualized



Orientation histogram of the cell



.67	.20
.17	.86

Length of the gradient vectors

3. Block normalization

After every cell’s histogram is calculated the blocks are normalized.

This means that a 2 cell × 2 cell block is selected and the histograms in the block are normalized with the sum of the histograms of the block.

The *stride* of the normalization is 1 (it means that the normalization is done with overlap, aka. one cell contributes in more than one block).

The feature vector:

- in each block: 36 x 1 long vector (concatenation of four 9-long vectors),
- in the entire image: *blocknumber* x 36 long giant vector.

Now please

download the ‘Lab 12’ code package

from the

submission system

Overview of the exercises:

Remark: we can be sure that all the input images has a size that is divisible by 8 and they are all grayscale.

Exercise 3

```
function norm_HOG = pmHOG_extractHOG(I)
  foreach block_of_image B
    foreach cell_of_block C
      [phi, mag] = pmHOG_gradient(C)
      hog = pmHOG_binner(phi, mag)
      ...
    end
    %normalize hogs_of_B
  end
end
```

Exercise 1

```
function [PHI, MAG] = pmHOG_gradient(I)
  ...
end
```

Exercise 2

```
function H = pmHOG_binner(PHI, MAG)
  ...
end
```

to be continued →

Overview of the exercises:

Exercise 4

```
function [HOG_differences, guilty_index] = pmHOG_findGuilty()
  % load evidence image, suspect database
  H_E = pmHOG_extractHOG(evidence)
  foreach suspect S
    H_S = pmHOG_extractHOG(S)
    % calculate difference between H_E and H_S
  end
  % find minimal difference
end
```

Exercise 1

Complete the function `pmHOG_gradient(I)`:

- in which compute the gradients of the cell and give back the pixelwise angle and magnitude data,
- the function has 3 parameters:
  - **Input1:** input image (**`I`**)
  - **Output1:** angle matrix (**`PHI`**)
  - **Output2:** magnitude matrix (**`MAG`**)

See the upcoming slides for hints.

Exercise 1 – hints

1. Convert the input image to double (with **`double`**).
2. Define the kernels as MATLAB row and column vectors.
3. Use **`imfilter`** to apply your kernels with the **`'replicate'`** and **`'same'`** parameters (as we want to pad the cell with replicated values and we want a result that has the same size as the input).
4. Compute the phase angle with the **`atan`** function. This returns the angles in radians, convert them to degrees using the **`rad2deg`** function.
5. Compute the magnitude of the vectors with the help of the **`sqrt`** function.
6. Test your function with **script** `test_gradient`.

Exercise 2

Complete the function `pmHOG_binner(PHI, MAG)`:

- in which realize the binning of the gradient values within a cell,
- the function has 3 parameters:
  - **Input1:** angle matrix (**`PHI`**)
  - **Input2:** magnitude matrix (**`MAG`**)
  - **Output**: cell histogram (**`H`**)

See the upcoming slides for hints.

Exercise 2 – hints

1. Initialize an empty 1×9 histogram vector. (`zeros`)
2. `for` each bin index
   a. Calculate the interval endpoints (minimum and maximum angle values).
   b. Create a logical mask of the phase matrix. Use the MATLAB logical indexing like
      `LOG_MASK = (PHI >= mini & PHI < maxi)`
   c. With the help of this logical matrix select those elements from the magnitude matrix for which the corresponding vector’s angle is in this bin. (So: index the magnitude matrix with the logical matrix. This returns the appropriate elements).
   d. Calculate the sum of the selected magnitude values and store them in the histogram at the position specified by the bin index.
3. Test your function with **script** `test_binner`.

Exercise 3

Complete the function `pmHOG_extractHOG(I)`:

- Divides the image into blocks, the blocks into cells, and calculates the cell histograms. This function also does the block normalization and feature-vector collection.
- It has 2 parameters:
  - **Input:** image matrix (**`I`**)
  - **Output**: block-normalized histograms of gradients in a matrix for every block (**`HOG`**)

See the upcoming slides for hints.

Exercise 3 – hints

1. Calculate the number of cells along the vertical and horizontal dimension of the image. **NOTE:** You can be sure that the image is grayscale and the image size is divisible by 8 (which is the cellsize). So the image is a H×W×1 matrix, where $H = 8h$ and $W = 8w$.
2. Initialize an empty (h-1)×(w-1)×36 histogram matrix (`zeros`). **NOTE:** h and w are the number of cells not the image height, width!
3. `for` each block --- *continued on the next slide*

Exercise 3 – hints continued

3. `for` each block
    - a. select the 16x16 pixel-sized sub-matrix from the grayscale image according to the block position (either with indexing, or with **`imcrop`**),
    - b. apply Gaussian filtering (**`imgaussfilt`** with standard deviation 0.1) --- the center of the block matters more,
    - c. initialize an empty 1x36 histogram vector (**`zeros`**) to the specific block,
    - d. `for` each cell (2x2 cells) inside the block
        - i. select the cell’s sub-matrix from the block,
        - ii. calculate the phase and magnitude values with the function you created in Exercise 1,
        - iii. calculate the histogram belonging to this cell with the function you created in Exercise 2,
        - iv. save the histogram into the histogram vector to the appropriate position,
    - e. after all the histograms are calculated inside one block, you have to do the block-normalization:
        - i. calculate the sum of the current block (the summation of the histogram vector),
        - ii. divide every element of the histogram vector by the calculated sum,
    - f. save the normalized histogram vector to the appropriate position of the histogram matrix (*save a 1D vector under a specific row&col position of a 3D matrix*), --- *continued on the next slide*

Exercise 3 – hints continued

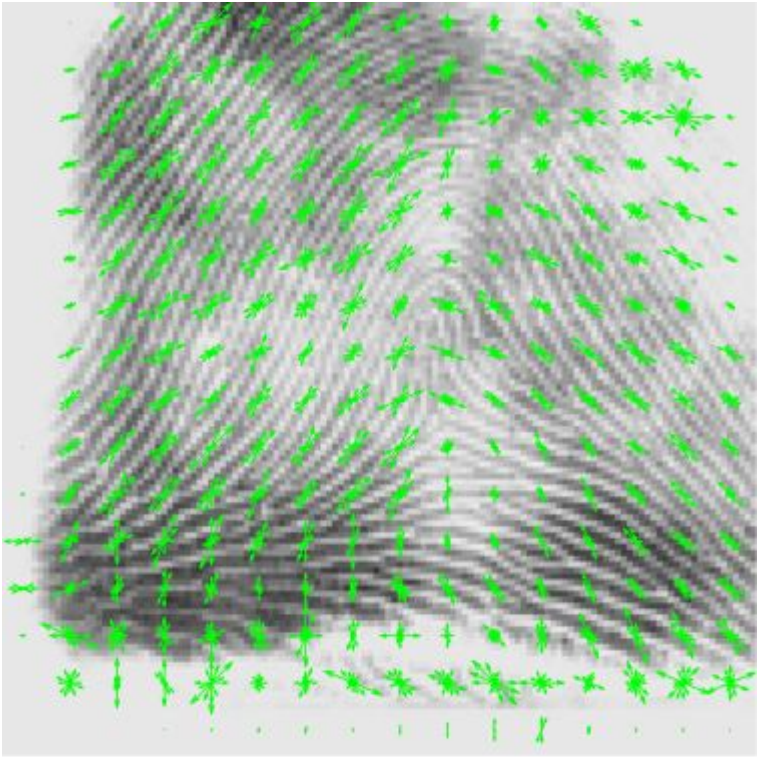
4. test your function with **script** `test_extractHOG`: this is a basic formal check,
5. test your function with **script** `test_extractHOG_2`.: this will visualize a part of the calculated descriptor over the sample image.

Please examine the results!

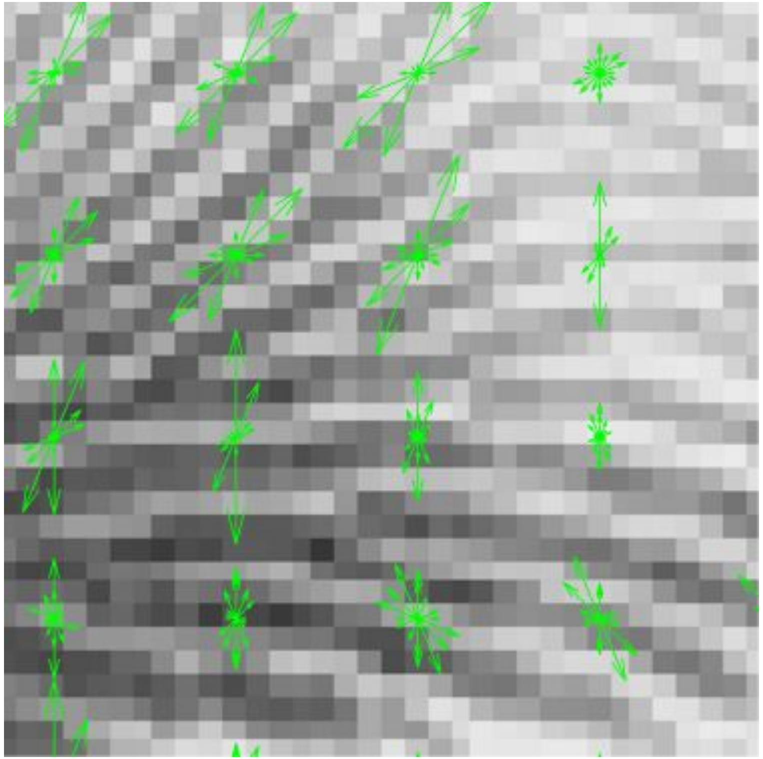
Zoom in, search for edges, compare HOG descriptors etc...

Exercise 3 – figures

Image and its HOG descriptors



Same image - zoomed in



Exercise 4

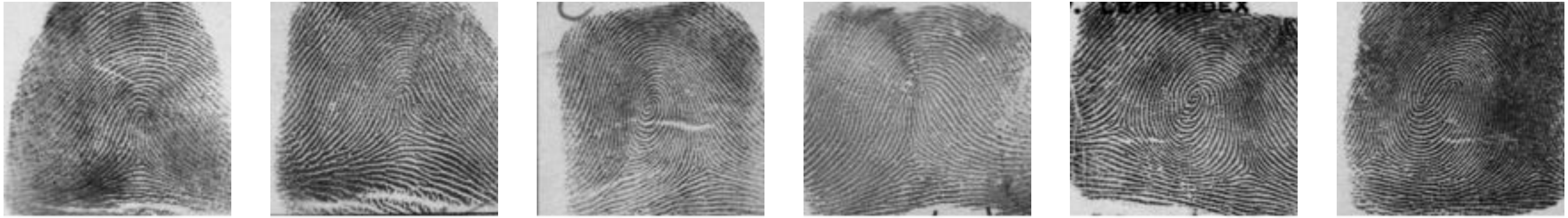
We have created a database from fingerprints. This database is called ‘suspects’ and it contains 10 images. The images are stored along the 3rd dimension. The database can be loaded in MATLAB with the following command:

```
load suspects.mat
```

You can inspect every image of the database using the appropriate indexing. Eg.

```
imshow(suspects(:,:,1));
```

plots the first fingerprint. Some of the fingerprints are also shown here:

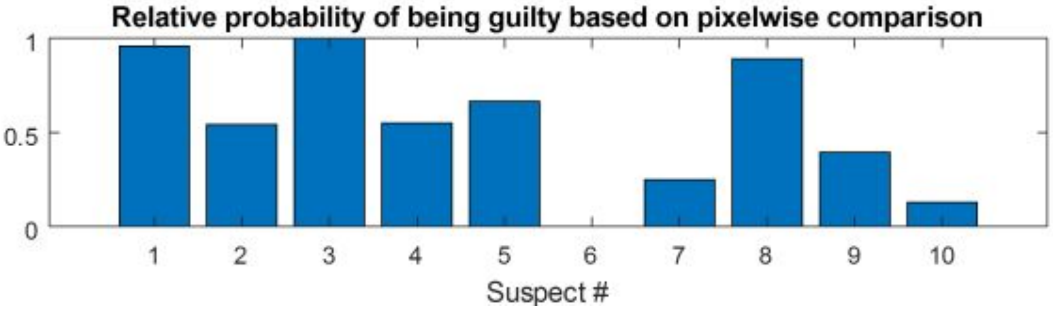


Exercise 4 continued

There is also an image (with which you have just met), it is the `'evidence.png'`.

This image shows a fingerprint. This image (I mean that the same image with the same pixel intensities) is not contained in the suspect database, however, the fingerprint can be found in the database but it is an another picture of it, so only the patterns matches, not the intensity levels.

Therefore it is impossible to match the evidence image with the suspects database based on pixel values. Such a comparison leads to the following result:



Exercise 4 continued

This result is not so impressive, Suspect 1, 3 and 8 are near equally likely to be guilty; there is no such a wide margin or something.
However, if you do the comparison based on the HOG descriptor differences the results are way more better.

Complete the function `pmHOG_findGuilty()`:

0. it has 0 input parameters, 2 outputs:
   a. **Output1:** summed difference-values of the HOG-descriptors (evidence vs suspects) (**`HOG_differences`**),
   b. **Output2:** guess about the culprit candidate (**`guilty_index`**),
1. load the suspect database and read in the evidence image,
2. calculate the HOG descriptor matrix of the evidence image (call it **`HOG_E`**),
3. for every suspect fingerprint…
   a. calculate the HOG descriptor matrix (call it **`HOG_S`**),
   b. calculate the difference of **`HOG_E`** and **`HOG_S`**: possible difference measure is the **`sum(sum(sum(abs(HOG_E-HOG_S))))`** (the sum of absolute differences),
   c. store this difference in a vector assigned to the suspect index,
4. find the index of the culprit (the lowest value in the difference vector - use **`min`**),
5. run your function with **script** `test_findGuilty`.

THE END