Lab 09

Basic Image Processing
Fall 2020

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

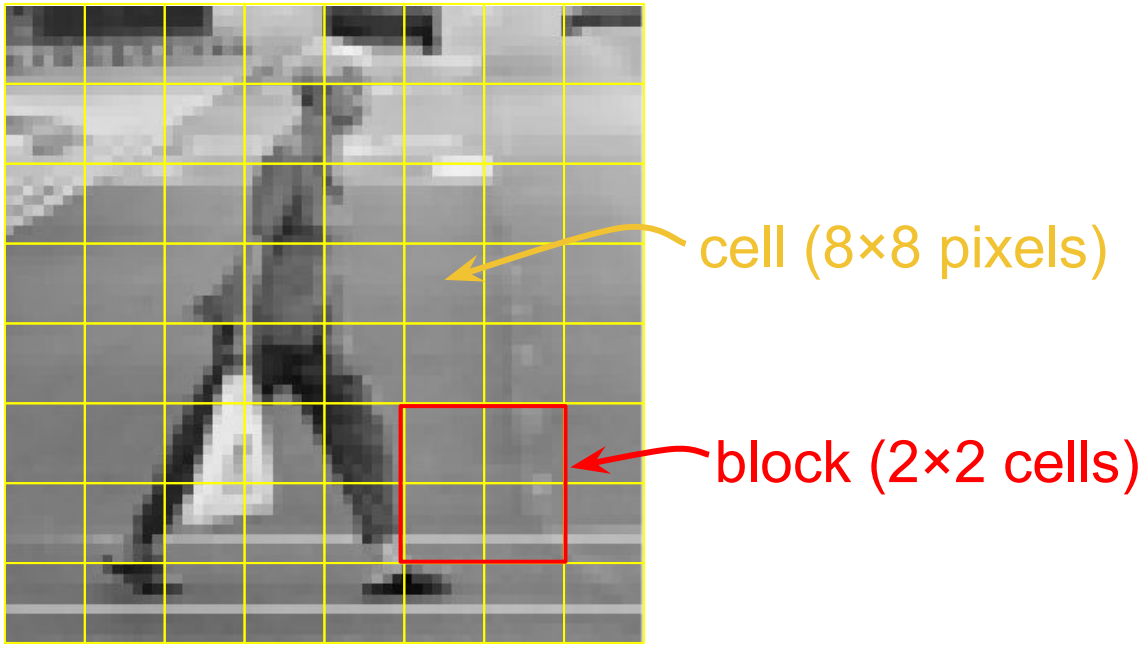
This algorithm is very similar to the original HOG method [1] but it contains some simplifications. The steps:

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

The upcoming slides contain 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 $\mathbf{G}$ 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 $\mathbf{G}$, $\mathbf{X}$, $\mathbf{Y}$ 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.

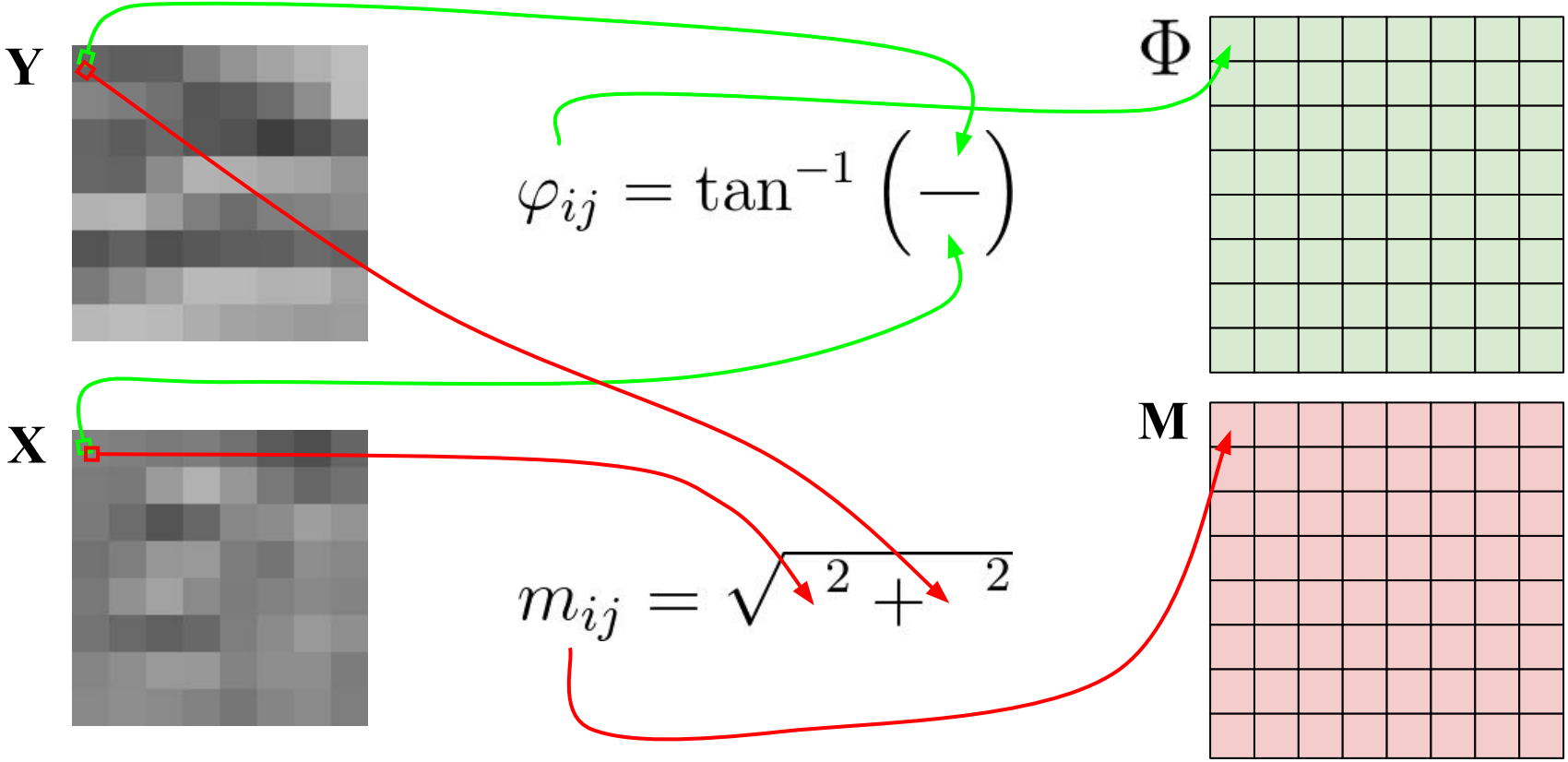
So! At every (i, j) point we know two values: x_{ij} and y_{ij}. These are the vertical and horizontal components of the gradient vector. The vector's angle is

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

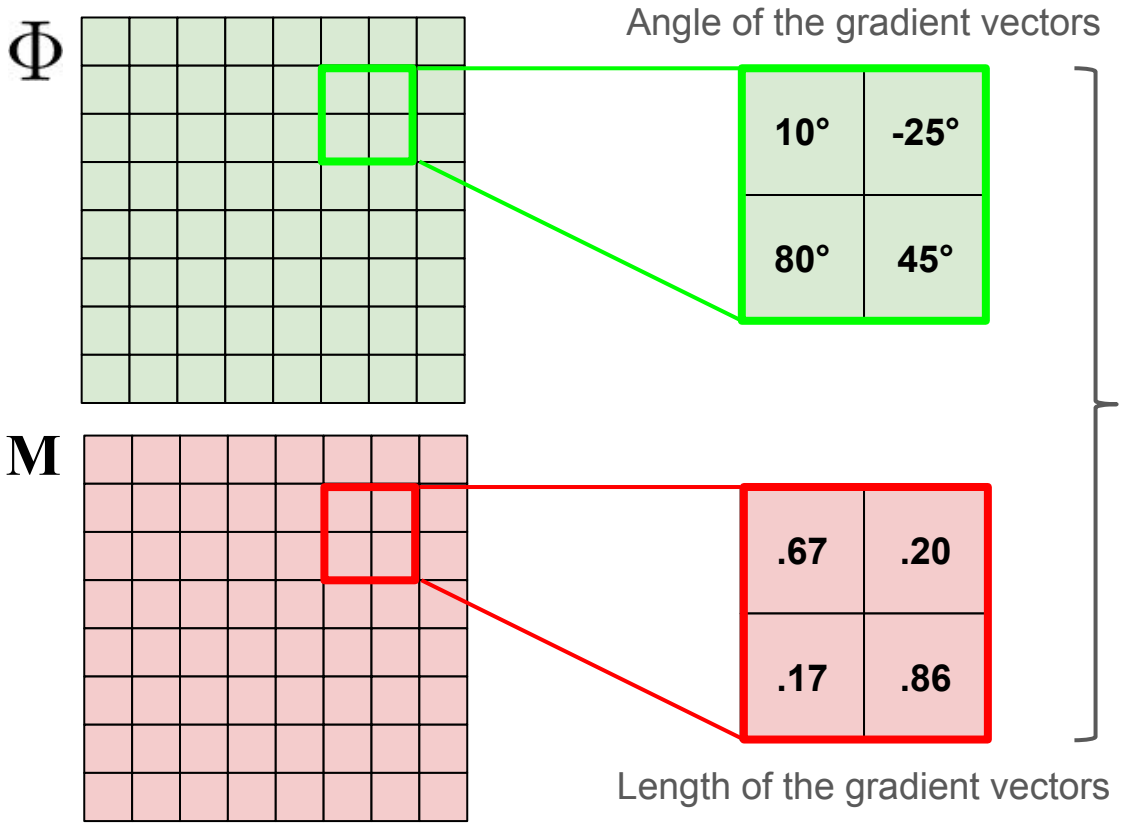
and the length of the gradient vector (magnitude) is

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

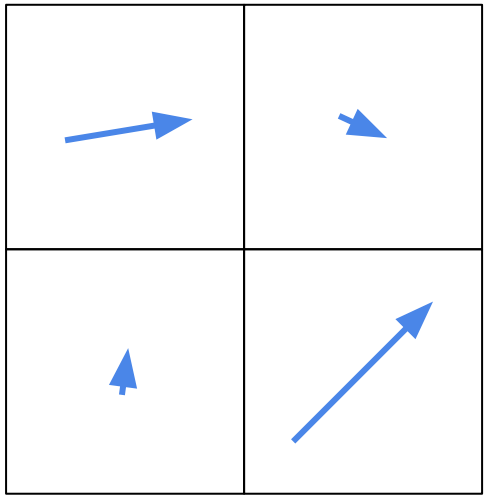
2. Orientation binning illustration part 1



2. Orientation binning illustration part 1



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.

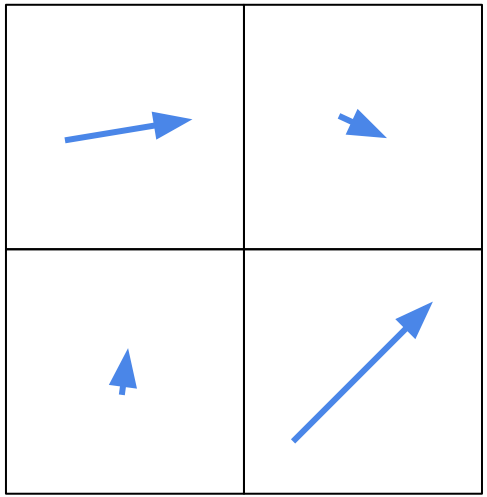
This procedure results in one orientation histogram per cell.

2. Orientation binning illustration part 2

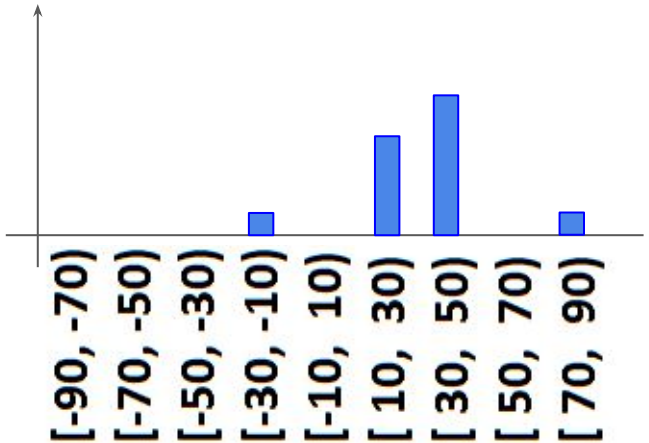
Angle of the gradient vectors

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).

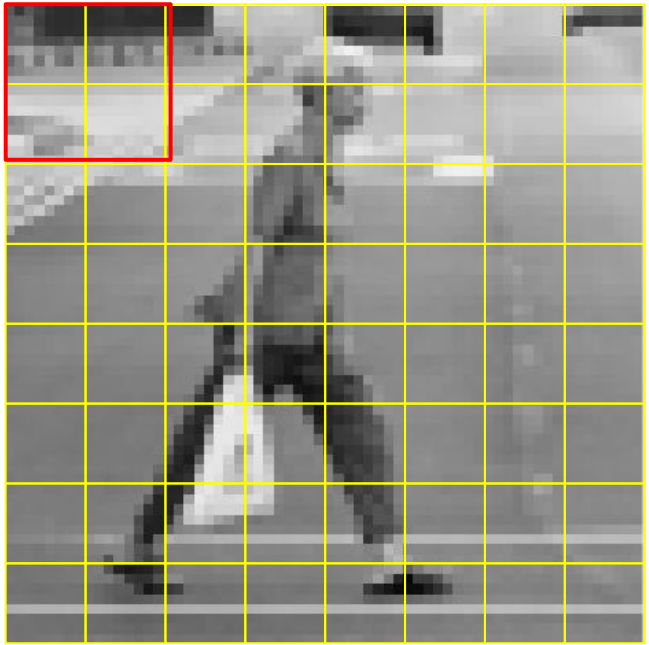
The feature vector:

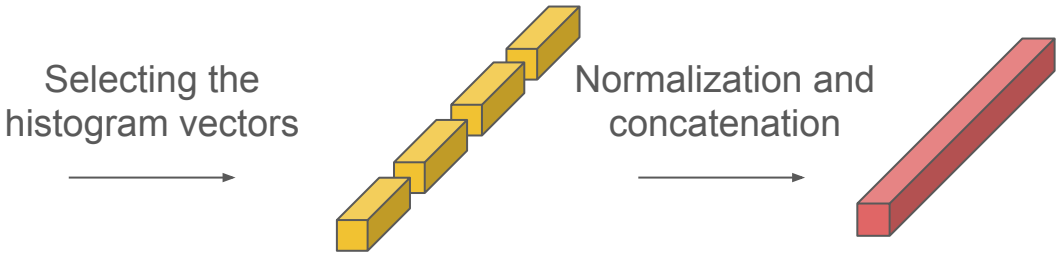
- in each block: 36 x 1 long vector (concatenation of four 9-long vectors),
- in the entire image: *blocknumber* x 36 long giant vector.

3. Block normalization

illustration

After the computation of all the cell histogram vectors, blocks are selected on the image. For each block, the four corresponding vectors are copied, normalized and concatenated into the block feature vector, which is the descriptor for the block.

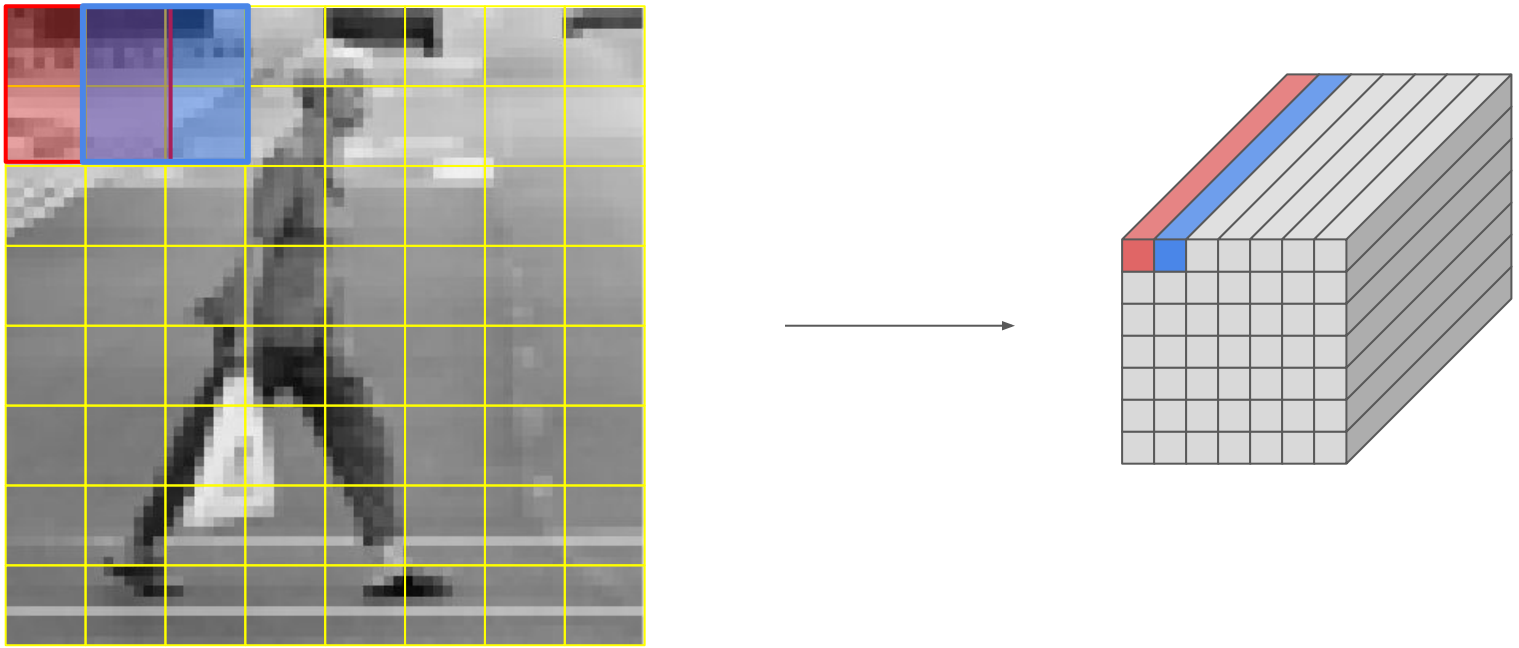




3. Block normalization

illustration

The blocks overlap, meaning that a typical cell is used to create multiple blocks. The number of blocks is $(h-1) \times (w-1)$ where h and w are the number of cells in rows and cols. Finally the image is described by a $(h-1) \times (w-1) \times 36$ HOG matrix.



Now please

download the ‘Lab 09’ code package

from the

moodle system

Overview of the exercises:

Exercise 1: implement the gradient computation for a single cell (8×8 pixels)

```
[PHI, MAG] = pmHOG_gradient(I)
```

Exercise 2: implement the orientation binning for a single cell (8×8 pixels)

```
[H] = pmHOG_binner(PHI, MAG)
```

Exercise 3: with the help of the previously implemented functions, implement the Poor Man's HOG descriptor computation function which computes the descriptors for the whole image

```
[HOG] = pmHOG_extractHOG(I)
```

Finally, in **Exercise 4** you will see an application of the HOG method.

Exercise 1

Complete the function `pmHOG_gradient` in which compute the gradients of the cell and return the pixelwise angle and magnitude data. The function has 3 parameters:

- **Input1:** input cell 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 the **script** `test1_gradient`.

Exercise 2

Complete the function `pmHOG_binner` 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 the **script** `test2_binner`.

Exercise 3

Complete the function `pmHOG_extractHOG` in which you have to divide the image into blocks, the blocks into cells, and calculate 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 $H \times W \times 1$ matrix, where $H = 8h$ and $W = 8w$.
2. Initialize the histogram matrix (`norm_HOG`) with zeros. The size of it is:

$$(h-1) \times (w-1) \times 36$$

NOTE: h and w are the number of cells not the image height, width!

Description continues on the next slide...

Exercise 3 – hints continued

3. For each block
   a. select the 16×16 pixel-sized sub-matrix from the grayscale image according to the block position
   b. apply Gaussian filtering (`imgaussfilt` with standard deviation 0.1) (the center of the block is more important).
   c. initialize an empty `feature_vector` for the specific block. This will store the concatenated orientation histogram vectors for the 4 cells within the current block.
   d. For each cell (2×2 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 describing this cell with the function you created in Exercise 2,
      iv. append the histogram to the `feature_vector`,
   e. after all the histograms are calculated inside one block, you have to do the block-normalization. Update the `feature_vector`, divide its elements by the sum of the vector:

```
feature_vector = feature_vector / sum(feature_vector)
```

   f. save the normalized `feature_vector` to the appropriate position of the `norm_HOG` matrix. (*Save a 1D vector under a specific row&col position of a 3D matrix.*)

Exercise 3 – testing

First, please test your function with **script** `test3_extractHOG`. This is a basic formal check.

Next, run the **script** `test4_visualizeHOG`. This will compute the HOG descriptors for an image and visualizes the calculated descriptor (plots the direction arrows).

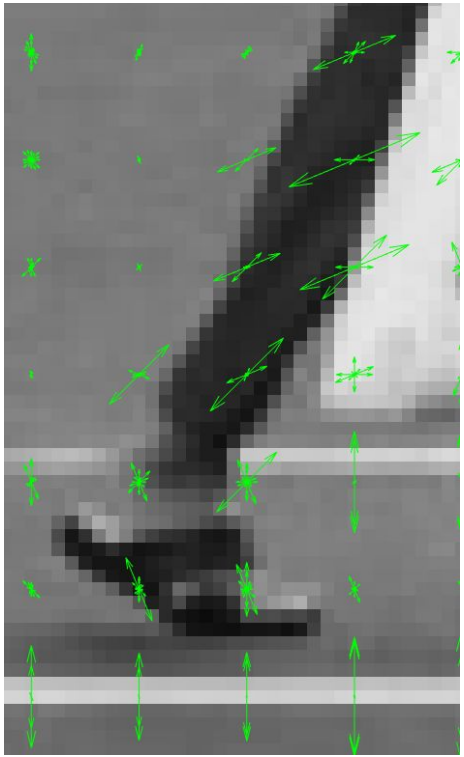
Please examine the results!

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

Exercise 3 – figures

Image and its HOG descriptors

Same image - zoomed in



Exercise 4

If you open the input folder you see (the grayscale version) of 5 different objects:



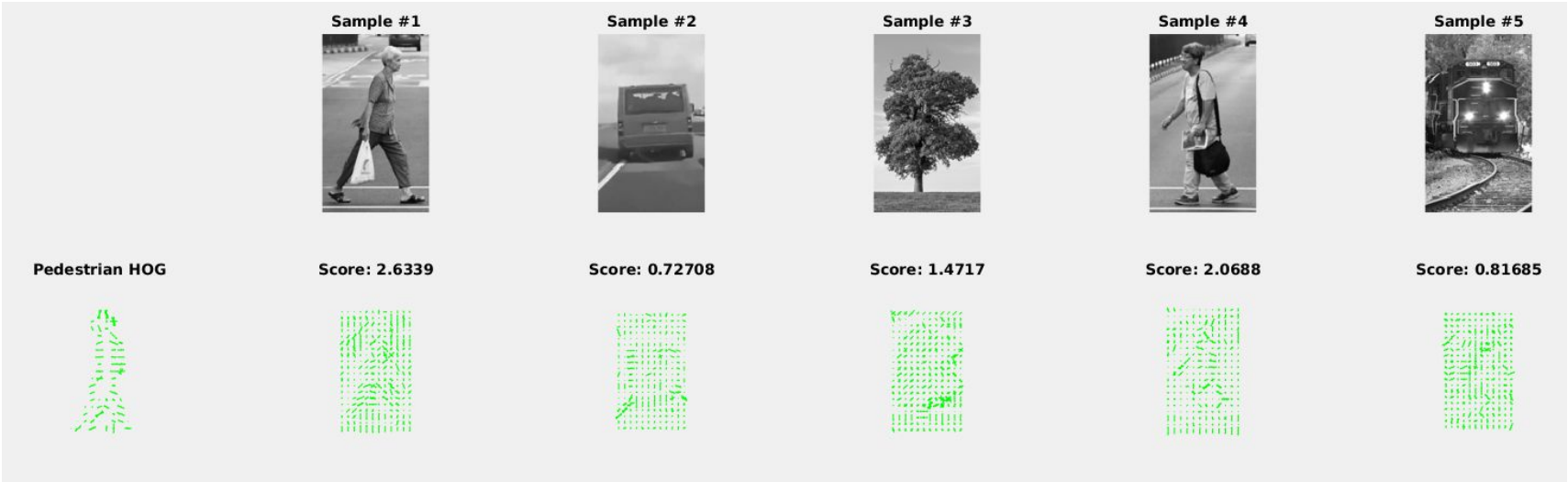


We want to detect **pedestrians**.

Exercise 4 continued

There is also a **mat file** `pedestrian_sample.mat`. This file contains a sample HOG descriptor showing a pedestrian walking in front of the camera.

Based on this sample we want to classify the five different inputs and tell whether they are pedestrians or not (with a confidence).



Exercise 4 continued

Complete the function `detectPedestrians` **in which** you have to implement a very simple HOG feature based pedestrian detector.

The function has two inputs (`I` is the grayscale image of the candidate, `sample_HOG` is a HOG model of a sample pedestrian) and one output (`score` is the score of the match — how likely is that the candidate is a pedestrian).

Inside the function:

1. Compute the HOG descriptor of the input image.
2. Do an elementwise multiplication between the HOG of the candidate and the HOG of the pedestrian model.
3. Get rid of the non significant values (keep only elements that are above 0.01).
4. Compute the score as the sum of all elements. Watch out, there are NaN values in the matrix, use `nansum` to eliminate them from the summation.

Finally, run script `test5_findPedestrians`. The script will compute the score of the 5 images and displays them in a figure.

Sample #1

Sample #2

Sample #3

Sample #5

Pedestrian HOG

Score: 2.6339

Score: 0.72708

Score: 1.4717

Score: 2.0688

Score: 0.81685

THE END