# Homework2

4190.408 001, Artificial Intelligence

October 1, 2025

## 1 Submission

- Assignment due: Friday, October 17, 2025 (11:59 p.m.)
- Individual assignment
- Submission through ETL. **Include both your codes and a PDF file that includes your output results and images.**
- You will complete HOG_ver1.py, which contains the following functions:
  - `extract_hog`
  - `get_differential_filter`
  - `filter_image`
  - `get_gradient`
  - `build_histogram`
  - `get_block_descriptor`
  - `face_detection`

  The code can be downloaded from ETL. If you can't, please contact our TAs.
- The function that does not comply with its specification will not be graded (no credit).
- The code must be run with Python 3 interpreter.
- Required python packages: `numpy`, `matplotlib`, and `opencv`.
  - numpy & matplotlib: `https://scipy.org/install.html`
  - OpenCV: `https://pypi.org/project/opencv-python/`
- You are not allowed to use any high-level python functions of image processing and computer vision, e.g., `cv2.filter2D`, `np.convolve`. Please consult with TA if you are not sure about the list of allowed functions.
- The following functions are allowed for use, as they are frequently asked about, but you are not required to use them: (`np.pad`, `np.linalg.norm`).
- We provide a visualization code for HOG. The resulting HOG descriptor must be able to be visualized with the provided code:
  `def visualize_hog(im, hog, cell_size, block_size)`

- We provide a visualization code for face detection. The detected faces must be able to be visualized with the provided code:
  `visualize_face_detection(I_target, bounding_boxes, bb_size)`
- Please place the code (HOG_ver1.py) as well as a one-page summary write-up with resulting visualization(in pdf format; if the write-up exceeds one-page, score can be deducted) into a folder named {STUDENT_NUMBER}_{NAME}, compress it into {STUDENT_NUMBER}_{NAME}.zip file, and submit. (ex. *202400000_GildongHong.zip*)
- This homework has been borrowed from Computer Vision Class of University of Minnesota. Special thanks to Professor Hyun Soo Park for allowing the use of this homework.

## 2 HOG

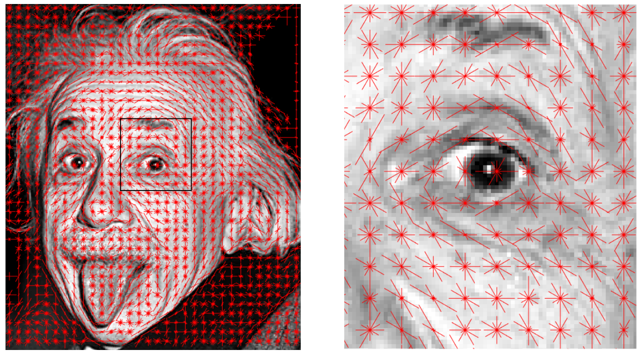


Figure 1: Histogram of oriented gradients. HOG feature is extracted and visualized for (a) the entire image and (b) the zoom-in image. The orientation and magnitude of the red lines represent the gradient components in a local cell.

In this assignment, you will implement a variant of HOG (Histogram of Oriented Gradients) in Python proposed by Dalal and Trigg [1] (2015 Longuet-Higgins Prize Winner). It had been a long-standing-top representation (until deep learning) for the object detection task with a deformable part model by combining with a SVM classifier [2]. Given an input image, your algorithm will compute the HOG feature and visualize it as shown in Figure 1 (the line directions are perpendicular to the gradient to show edge alignment). The orientation and magnitude of the red lines represent the gradient components in a local cell.

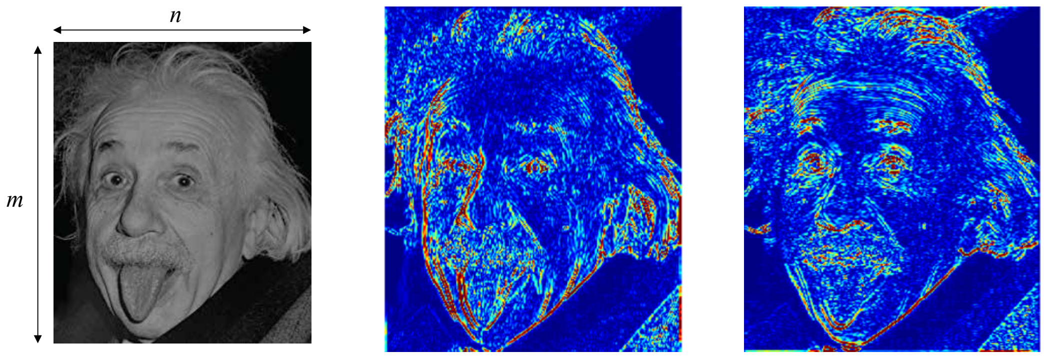
```
def extract_hog(im):
  ...
  return hog
```

**Input:** input gray-scale image with `uint8` format.
**Output:** HOG descriptor.
**Description:** You will compute the HOG descriptor of input image `im`. The pseudo-code can be found below:

**Algorithm 1** HOG

Convert the gray-scale image to `float` format and normalize to range [0, 1].
Get differential images using `get_differential_filter` and `filter_image`
Compute the gradients using `get_gradient`
Build the histogram of oriented gradients for all cells using `build_histogram`
Build the descriptor of all blocks with normalization using `get_block_descriptor`
Return a long vector (`hog`) by concatenating all block descriptors.

## 2.1 Image filtering




(a) Input image

(b) Differential along x direction

(c) Differential along y direction

Figure 2: (a) Input image dimension. (b-c) Differential image along $x$ and $y$ directions.

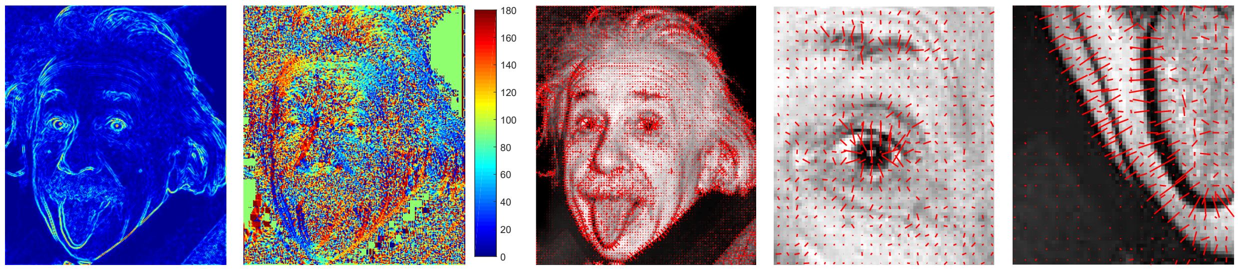
```
def get_differential_filter():
  ...
  return filter_x, filter_y
```

**Input:** none.
**Output:** `filter_x` and `filter_y` are 3×3 filters that differentiate along $x$ and $y$ directions, respectively.
**Description:** You will compute the gradient by differentiating the image along $x$ and $y$ directions. This code will output the differential filters.

```
def filter_image(im, filter):
  ...
  return im_filtered
```

**Input:** `im` is the grayscale $m \times n$ image (Figure 2(a)) converted to float format and `filter` is a filter ($k \times k$ matrix)
**Output:** `im_filtered` is $m \times n$ filtered image. You may need to pad zeros on the boundary on the input image to get the same size filtered image.
**Description:** Given an image and filter, you will compute the filtered image. Given the two functions above, you can generate differential images by visualizing the magnitude of the filter response as shown in Figure 2(b) and 2(c).

## 2.2 Gradient Computation



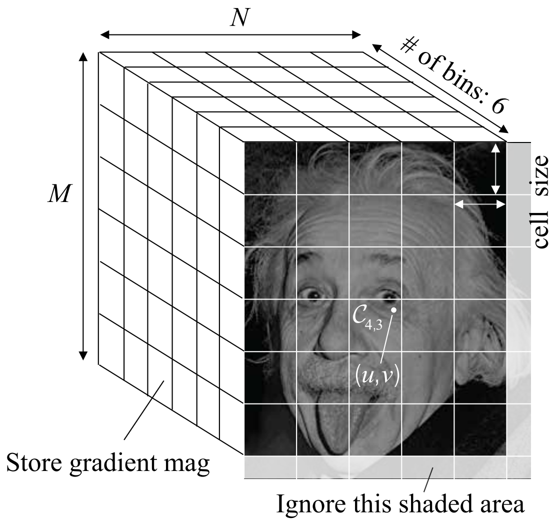

(a) Magnitude (b) Angle (c) Gradient (d) Zoomed eye (e) Zoomed neck

Figure 3: Visualization of (a) magnitude and (b) orientation of image gradients. (c-e) Visualization of gradients at every 3rd pixel (the magnitudes are re-scaled for illustrative purposes.).

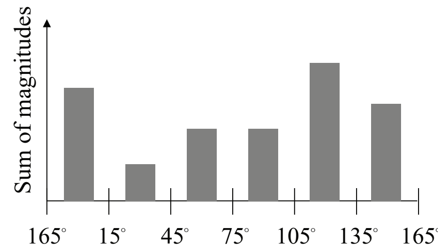
```
def get_gradient(im_dx, im_dy):
    ...
    return grad_mag, grad_angle
```

**Input:** `im_dx` and `im_dy` are the $x$ and $y$ differential images (size: $m \times n$).
**Output:** `grad_mag` and `grad_angle` are the magnitude and orientation of the gradient images (size: $m \times n$). Note that the range of the angle should be $[0, \pi)$, i.e., unsigned angle ($\theta == \theta + \pi$).
**Description:** Given the differential images, you will compute the magnitude and angle of the gradient. Using the gradients, you can visualize and have some sense of the image, i.e., the magnitude of the gradient is proportional to the contrast (edge) of the local patch, and the orientation is perpendicular to the edge direction as shown in Figure 3.

## 2.3 Orientation Binning




(a) ori_histo




(b) Histogram per cell

Figure 4: (a) Histogram of oriented gradients can be built by (b) binning the gradients to the corresponding bin.

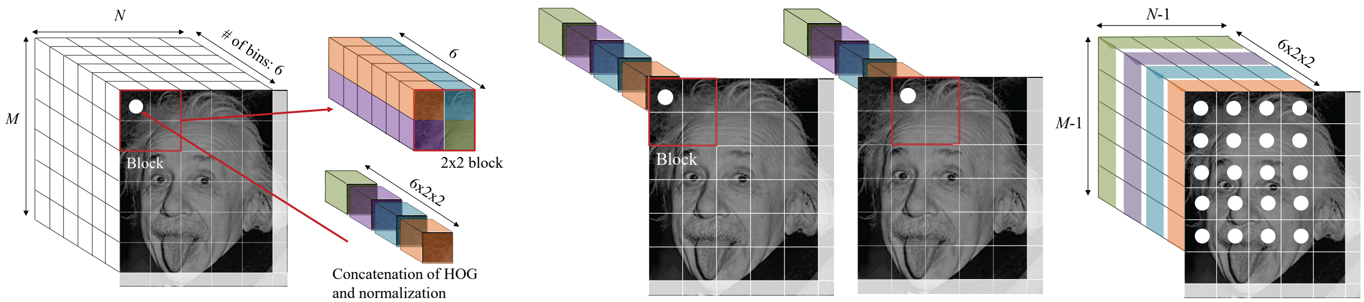
```
def build_histogram(grad_mag, grad_angle, cell_size):
  ...
  return ori_histo
```

**Input:** `grad_mag` and `grad_angle` are the magnitude and orientation of the gradient images (size: $m \times n$); `cell_size` is the size of each cell, which is a positive integer.
**Output:** `ori_histo` is a 3D tensor with size $M \times N \times 6$ where $M$ and $N$ are the number of cells along $y$ and $x$ axes, respectively, i.e., $M = \lfloor m/\texttt{cell\_size} \rfloor$ and $N = \lfloor n/\texttt{cell\_size} \rfloor$ where $\lfloor \cdot \rfloor$ is the round-off operation as shown in Figure 4(a).
**Description:** Given the magnitude and orientation of the gradients per pixel, you can build the histogram of oriented gradients for each cell.

$$\texttt{ori\_histo}(i,j,k) = \sum_{(u,v)\in\mathcal{C}_{i,j}} \texttt{grad\_mag}(u,v) \quad \text{if } \texttt{grad\_angle}(u,v) \in \theta_k$$

where $\mathcal{C}_{i,j}$ is a set of $x$ and $y$ coordinates within the $(i,j)$ cell, and $\theta_k$ is the angle range of each bin, e.g., $\theta_1 = [165°, 180°) \cup [0°, 15°)$, $\theta_2 = [15°, 45°)$, $\theta_3 = [45°, 75°)$, $\theta_4 = [75°, 105°)$, $\theta_5 = [105°, 135°)$, and $\theta_6 = [135°, 165°)$. Therefore, `ori_histo(i,j,:)` returns the histogram of the oriented gradients at $(i,j)$ cell as shown in Figure 4(b). Using the `ori_histo`, you can visualize HOG per cell where the magnitude of the line is proportional to the histogram, as shown in Figure 1. Typical `cell_size` is 8.

### 2.4 Block Normalization




(a) Block descriptor (b) Block overlap with stride 1

Figure 5: HOG is normalized to account for illumination and contrast to form a descriptor for a block. (a) HOG within the (1,1) block is concatenated and normalized to form a long vector of size 24. (b) This applies to the rest block with overlap and stride 1 to form the normalized HOG.

```
def get_block_descriptor(ori_histo, block_size):
  ...
  return ori_histo_normalized
```

**Input:** `ori_histo` is the histogram of oriented gradients without normalization. `block_size` is the size of each block (e.g., the number of cells in each row/column), which is a positive integer.
**Output:** `ori_histo_normalized` is the normalized histogram (size: $(M-(\texttt{block\_size}-1)) \times (N-(\texttt{block\_size}-1)) \times (6 \times \texttt{block\_size}^2)$.
**Description:** To account for changes in illumination and contrast, the gradient strengths must be locally normalized, which requires grouping the cells together into larger, spatially connected blocks (adjacent cells). Given the histogram of oriented gradients, you apply $L_2$ normalization as follow:

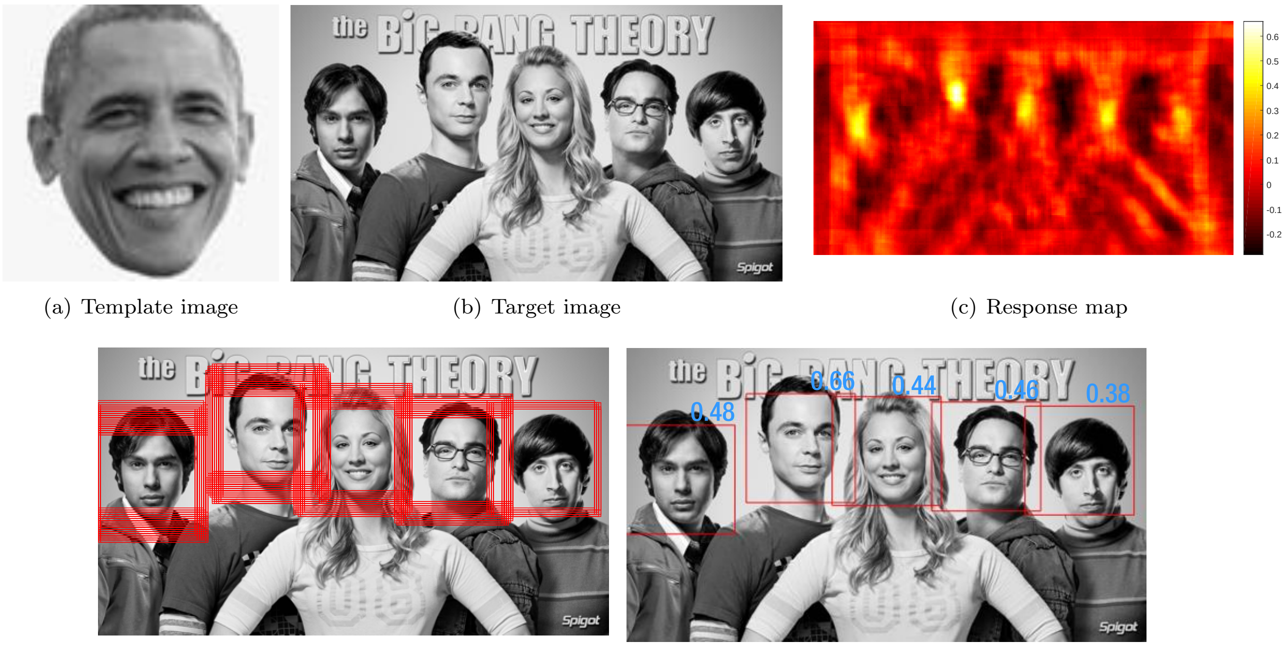
1. Build a descriptor of the first block by concatenating the HOG within the block. You can use `block_size`=2, i.e., $2 \times 2$ block will contain $2 \times 2 \times 6$ entries that will be concatenated to form one long vector as shown in Figure 6(c).
2. Normalize the descriptor as follows:

$$\hat{h}_i = \frac{h_i}{\sqrt{\sum_i h_i^2 + e^2}} \tag{1}$$

   where $h_i$ is the $i^{\text{th}}$ element of the histogram and $\hat{h}_i$ is the normalized histogram. $e$ is the normalization constant to prevent division by zero (e.g., $e = 0.001$).
3. Assign the normalized histogram to `ori_histo_normalized`(1,1) (white dot location in Figure 6(c)).
4. Move to the next block `ori_histo_normalized`(1,2) with the stride 1 and iterate 1-3 steps above.

The resulting `ori_histo_normalized` will have the size of $(M-1) \times (N-1) \times 24$.

## 2.5 Application: Face Detection




(a) Template image (b) Target image (c) Response map

(d) Thresholding (e) Non-maximum suppression

Figure 6: You will use (a) a single template image to detect faces in (b) the target image using HOG descriptors. (c) HOG descriptors from the template and target image patches can be compared by using the measure of normalized cross-correlation (NCC). (d) Thresholding on the NCC score will produce many overlapping bounding boxes. (e) Correct bounding boxes for faces can be obtained by using non-maximum suppression.

Using the HOG descriptor, you will design a face detection algorithm. You can download the template and target images from ETL. If you have any issues, don't hesitate to contact our TAs.

```
def face_recognition(I_target, I_template):
  ...
  return bounding_boxes
```

**Input:** `I_target` is the image that contains multiple faces. `I_template` is the template face image that will be matched to the image to detect faces.

**Output:** `bounding_boxes` is $n \times 3$ array that describes the $n$ detected bounding boxes. Each row of the array is $[x_i, y_i, s_i]$ where $(x_i, y_i)$ is the left-top corner coordinate of the $i^{\text{th}}$ bounding box, and $s_i$ is the normalized cross-correlation (NCC) score between the bounding box patch and the template:

$$s = \frac{\overline{\mathbf{a}} \cdot \overline{\mathbf{b}}}{\|\overline{\mathbf{a}}\| \|\overline{\mathbf{b}}\|} \tag{2}$$

where $\mathbf{a}$ and $\mathbf{b}$ are two normalized descriptors, i.e., zero mean:

$$a_i = a_i - \widetilde{\mathbf{a}} \tag{3}$$

where $a_i$ is the $i^{\text{th}}$ element of $\mathbf{a}$, and $a_i$ is the $i^{\text{th}}$ element of the HOG descriptor. $\widetilde{\mathbf{a}}$ is the mean of the HOG descriptor.

**Description:** You will use thresholding and **non-maximum suppression** with IoU 50% to localize the faces. You may use `visualize_face_detection(I_target, bounding_boxes, bb_size)` to visualize your detection. Please read this article if you do not know about IoU or NMS.

Non-Maximum Suppression: `https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c`
IoU: `https://medium.com/analytics-vidhya/iou-intersection-over-union-705a39e7acef`.

## References


[1] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005.

[2] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. *TPAMI*, 2010.