

CSCI 5561: Project #1

Histogram of Oriented Gradients (HOG)

1 Submission

- Assignment due: **Thursday, September 25, 2025 (11:59pm)**
- Individual assignment
- Please submit your assignment to the Gradescope **Autograder**.
- You will complete `p1.py` that contains the following functions:
 - `get_differential_filter`
 - `filter_image`
 - `get_gradient`
 - `build_histogram`
 - `get_block_descriptor`
 - `extract_hog`
 - `face_detection`
 - [Bonus] `face_detection_bonus`

`p1.py` is attached with the Canvas assignment for you to download.

- Any function that does not comply with its specification will not be graded (no credit).
- The code must be run with the Python 3 interpreter.
- Required Python packages: `numpy`, `matplotlib`, and `PIL`.
 - `numpy` & `matplotlib`: <https://scipy.org/install.html>
 - `PIL`: <https://pypi.org/project/pillow/>
- You are not allowed to use any high-level Python functions of image processing and computer vision, such as functions from OpenCV (e.g., `cv2.filter2D`). Please consult with the TA if you are not sure whether a function is allowed.
- We provide a visualization code for HOG. The resulting HOG descriptor must be able to be visualized with the provided code:

```
def visualize_hog(image, hog, cell_size, block_size, num_bins)
```
- We provide a visualization code for face detection. The detected faces must be able to be visualized with the provided code:

```
def visualize_face_detection(I_target, bounding_boxes, box_size)
```

CSCI 5561: Project #1

Histogram of Oriented Gradients (HOG)

2 HOG

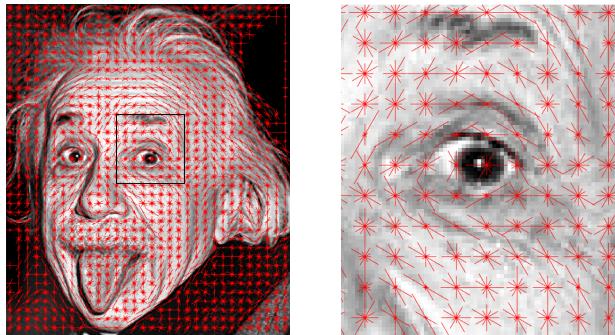


Figure 1: Histogram of oriented gradients. HOG feature is extracted and visualized for (a) the entire image and (b) zoomed-in image. The orientation and magnitude of the red lines represent the gradient components in a local cell.

In this assignment, you will implement a variant of HOG (Histogram of Oriented Gradients) in Python proposed by Dalal and Triggs [1] (2015 Longuet-Higgins Prize Winner). It had been long standing top representation (until deep learning) for the object detection task with a deformable part model by combining with an SVM classifier [2]. Given an input image, your algorithm will compute the HOG feature and visualize 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. You should begin by implementing the functions in Sections 2.1 through 2.4.

```
def extract_hog(image, cell_size=8, block_size=2):  
    ...  
    return hog
```

Input: A grayscale image with `uint8` format.

Output: HOG descriptor.

Description: You will compute the HOG descriptor of input image. The pseudo-code can be found below:

Algorithm 1 HOG

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

CSCI 5561: Project #1

Histogram of Oriented Gradients (HOG)

2.1 Image filtering

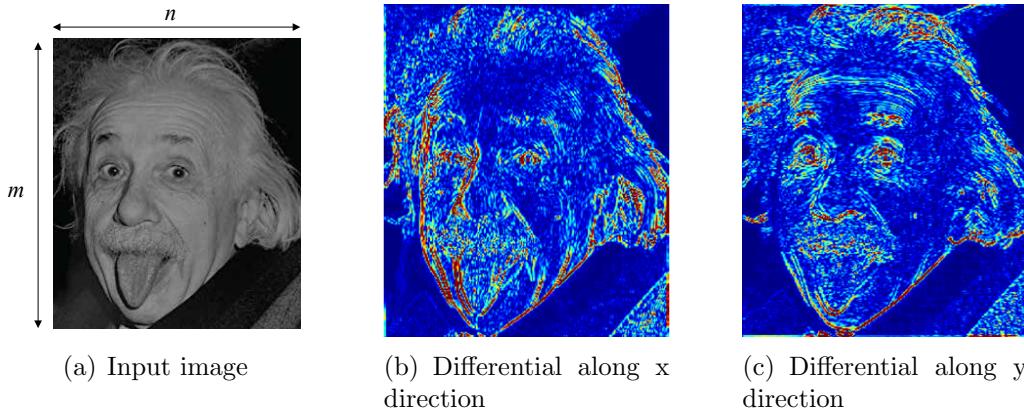


Figure 2: (a) Input image dimension. (b-c) Differential image along x and y directions.

```
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(image, filter):
    ...
    return image_filtered
```

Input: `image` is the grayscale $m \times n$ image (Figure 2(a)) converted to float format and `filter` is a filter ($k \times k$ matrix)

Output: `image_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).

CSCI 5561: Project #1

Histogram of Oriented Gradients (HOG)

2.2 Gradient Computation

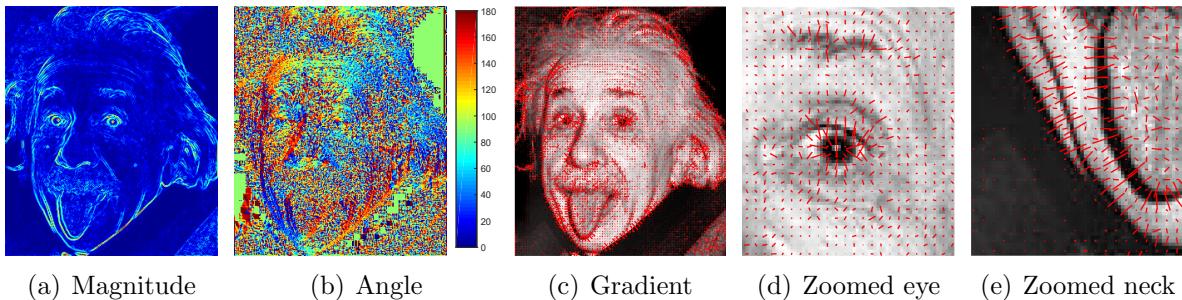


Figure 3: Visualization of (a) magnitude and (b) orientation of image gradients. (c-e) Visualization of gradients at every 3rd pixel (the magnitudes are re-scaled for illustrative purposes.).

```
def get_gradient(image_dx, image_dy):  
    ...  
    return grad_mag, grad_angle
```

Input: `image_dx` and `image_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 with 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.

CSCI 5561: Project #1

Histogram of Oriented Gradients (HOG)

2.3 Orientation Binning

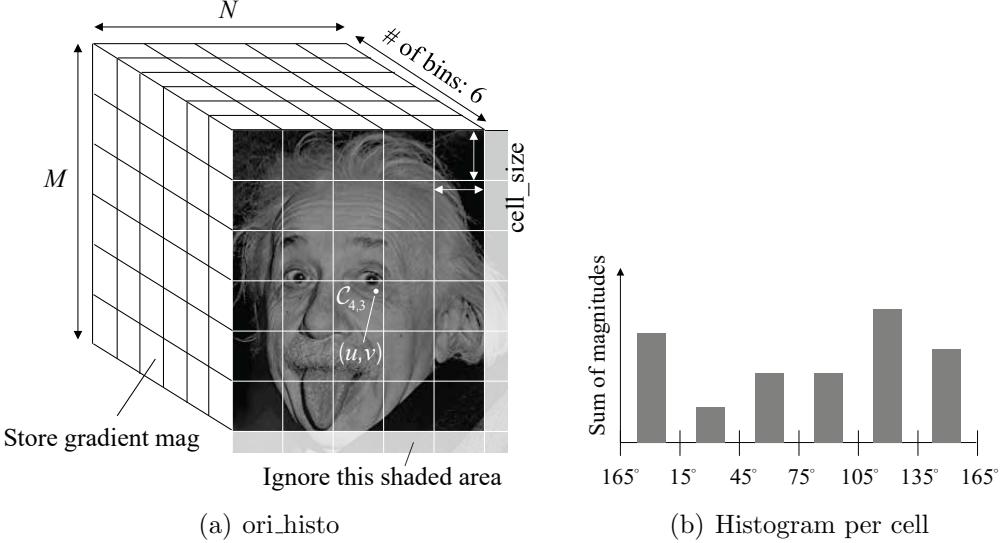


Figure 4: (a) Histogram of oriented gradients can be built by (b) binning the gradients to corresponding bin.

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

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_hist` 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., $M = \lfloor m/\text{cell_size} \rfloor$ and $N = \lfloor n/\text{cell_size} \rfloor$ where $\lfloor \cdot \rfloor$ is the floor 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.

$$\text{ori_histo}(i, j, k) = \sum_{(u, v) \in C_{i,j}} \text{grad_mag}(u, v) \quad \text{if } \text{grad_angle}(u, v) \in \theta_k \quad (1)$$

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

CSCI 5561: Project #1

Histogram of Oriented Gradients (HOG)

2.4 Block Normalization

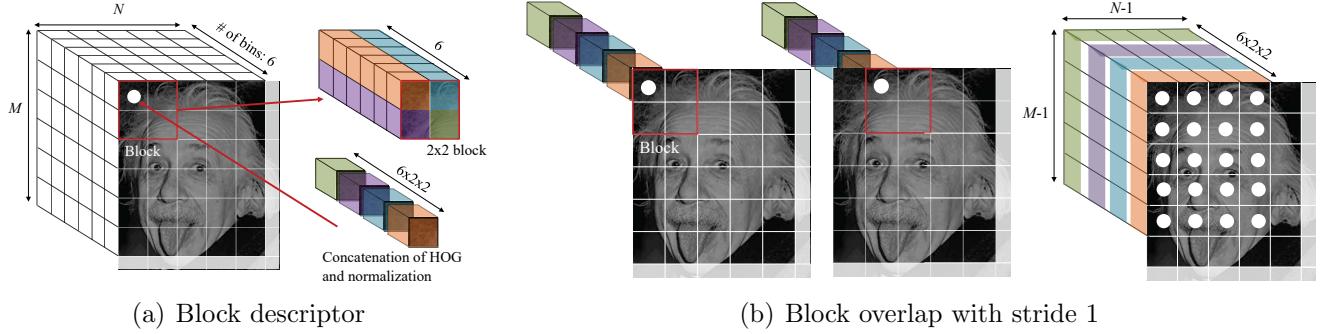


Figure 5: HOG is normalized to account for illumination and contrast to form a descriptor for a block. (a) HOG within (1,1) block is concatenated and normalized to form a long vector of size 24. (b) This applies to the rest block with overlap and stride 1 to form the normalized HOG.

```
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: $(M - (\text{block_size} - 1)) \times (N - (\text{block_size} - 1)) \times (6 \times \text{block_size}^2)$).

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 follows:

1. Build a descriptor of the first block by concatenating the HOG within the block. You can use `block_size=2`, i.e., 2×2 block will contain $2 \times 2 \times 6$ entries that will be concatenated to form one long vector as shown in Figure 5(a).
2. Normalize the descriptor as follow:

$$\hat{h}_i = \frac{h_i}{\sqrt{\sum_i h_i^2 + e^2}} \quad (2)$$

where h_i is the i^{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 5(a)).
4. Move to the next block `ori_histo_normalized(1,2)` with stride 1 and iterate 1-3 steps above.

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

CSCI 5561: Project #1

Histogram of Oriented Gradients (HOG)

2.5 Application: Face Detection

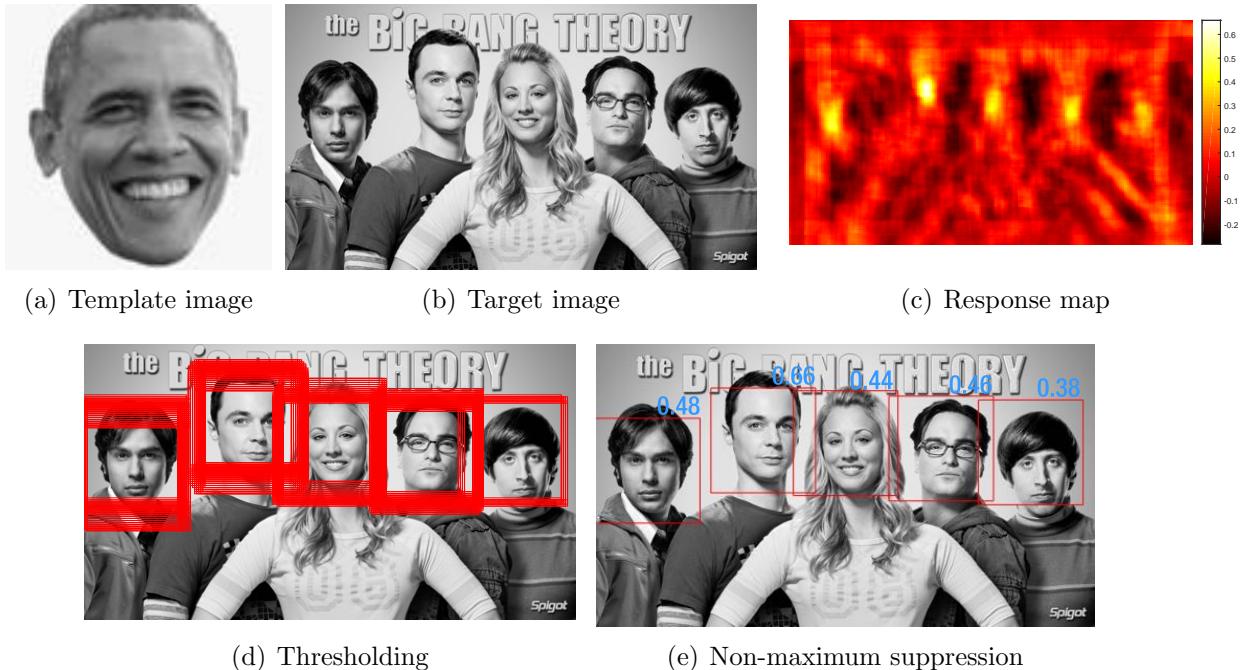


Figure 6: You will use (a) a single template image to detect faces in (b) the target image using HOG descriptors. (c) HOG descriptors from the template and target image patches can be compared by using the measure of normalized cross-correlation (NCC). (d) Thresholding on NCC score will produce many 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. The template and target images can be found in `assets.zip` on the Canvas assignment.

```
def face_detection(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^{th} bounding box, and s_i is the normalized cross-correlation (NCC) score between the bounding box patch and the template:

$$s = \frac{\bar{\mathbf{a}} \cdot \bar{\mathbf{b}}}{\|\bar{\mathbf{a}}\| \|\bar{\mathbf{b}}\|} \quad (3)$$

CSCI 5561: Project #1

Histogram of Oriented Gradients (HOG)

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

$$\bar{a}_i = a_i - \tilde{\mathbf{a}} \quad (4)$$

where \bar{a}_i is the i^{th} element of \mathbf{a} , and a_i is the i^{th} element of the HOG descriptor. $\tilde{\mathbf{a}}$ 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

```
def visualize_face_detection(I_target, bounding_boxes, box_size)
```

to visualize your detection.

CSCI 5561: Project #1

Histogram of Oriented Gradients (HOG)

2.6 [Bonus] Face Detection with Varying Sizes

Extend your HOG-based face detector to handle faces at multiple scales. You will search over an image pyramid and perform detection at each scale, followed by non-maximum suppression (NMS) across all scales. The test image is provided as `bonus.jpg` in `assets.zip` on the Canvas assignment.

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

Input: `I_target` is the image that contains multiple faces. `I_template` is the template face image to be matched. You may also experiment with different templates if desired.

Output: `bounding_boxes` is an $n \times 3$ 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^{th} bounding box and s_i is the normalized cross-correlation (NCC) score.

Submission (by Email Only)

Do **not** submit the bonus to Gradescope. Instead, email the following to `lee04484@umn.edu`:

- Your script `p1.py`
- Your detection result image `bonus_result.jpg`

Email subject: [CSCI5561] P1 Bonus - <Your Name>

Bonus credit: 2 points.

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.