UNIVERSITY OF TEXAS AT ARLINGTON
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

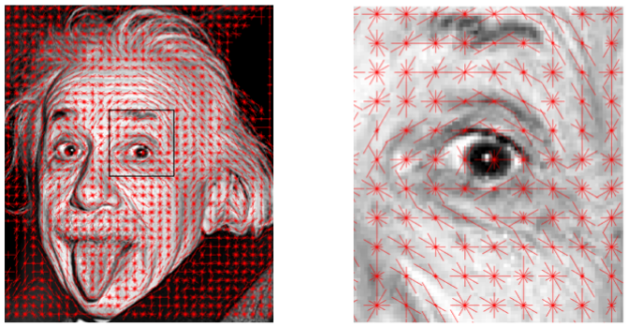
6367 **COMPUTER VISION** **SPRING 2020**

ASSIGNMENT 4 (100 POINTS)
ASSIGNED: 3/17/2020 DUE: 4/2/2020

This assignment constitutes 10% of the course grade. You must work on it individually and are required to submit a PDF report along with the MATLAB scripts described below. The use of high-level built-in MATLAB functions for image processing and computer vision (e.g. `imfilter` and `conv2`) are not permitted except for I/O functions such as `imread`, `imwrite`, and `imagesc`.

The objective of this assignment is to implement a variant of the histogram of oriented gradients (HOG) descriptor proposed by Dalal and Trigg [1]. The HOG descriptor was the state of the art for feature representation until the advent of deep learning. It has been used for the task of object detection with deformable part models by combining the descriptor with an SVM classifier [2].

1 Histogram of Oriented Gradients



(a) (b)

Figure 1: Histogram of oriented gradients (HOG) features extracted and visualized for the entire image (a) along with a zoomed in region (b). The orientation and magnitude of the red lines represents the gradient components in a local cell.

Given an input image, you will implement an algorithm to compute the HOG features and visualize them as shown in Figure 1. Note that the line directions are perpendicular to the gradient to show the edge alignments. The orientation and magnitude of the red lines corresponds to the gradient components in a local cell. The pseudocode for computing the HOG descriptor is shown in Algorithm 1 and the implementation instructions are provided in Sections 2 - 5.

Algorithm 1 HOG

Input: Grayscale image in `uint8` format
Output: HOG descriptor

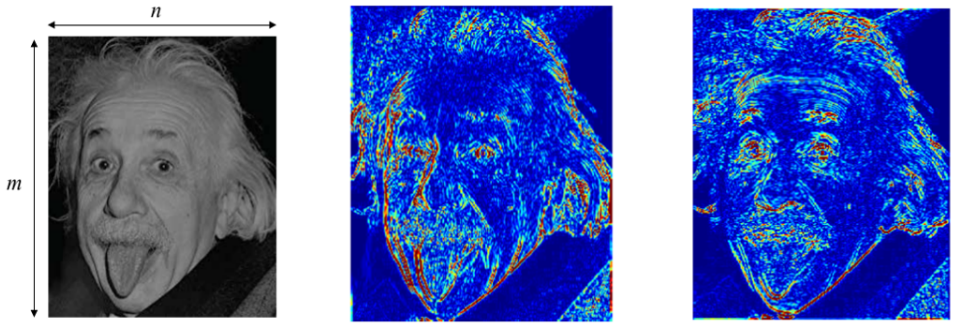
1: Convert the grayscale image to double format
2: Compute the differential images using `filter_image`
3: Compute the gradients using `get_gradients`
4: Build the histogram of oriented gradients for all cells using `build_histogram`
5: Build the normalized descriptors for all blocks using `get_block_descriptor`
6: Return a long vector (HOG) by concatenating all the block descriptors

Problem 1 (5 points)

Write a MATLAB function, `[hog] = HOG(im)`, that implements Algorithm 1. The input, `im`, is a grayscale image and the output, `hog`, is a long vector of concatenated block descriptors.

Submission Instructions: *Submit a MATLAB script, hog.m, that performs the operations in Algorithm 1. Each of these operations is explained, in detail, in Sections 2 - 5. Compute the HOG descriptor for the included image 'einstein.jpg' and two additional images of your choice.*

2 Image Filtering



(a) Input image. (b) Differential along the x direction. (c) Differential along the y direction.

Figure 2: An input image (a), and the differential images along the x (b) and the y (c) directions.

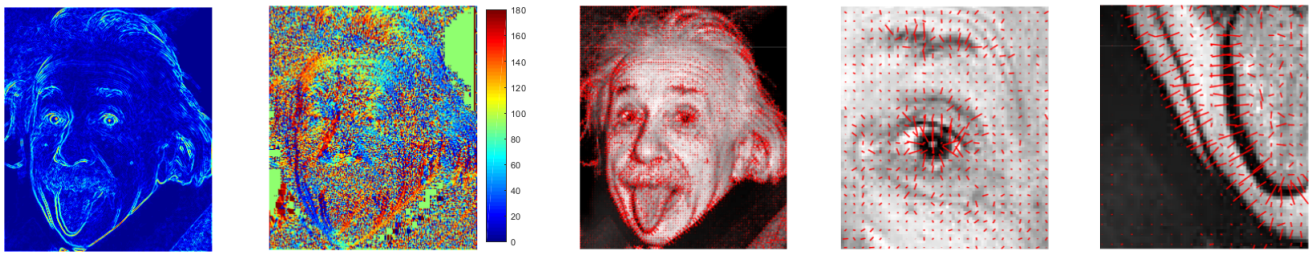
Problem 2 (20 points)

Write a MATLAB function, `[im_dx, im_dy] = filter_image(im)`, that computes the gradient of the input image by differentiating the image along the x and y directions. The input, `im`, is a grayscale $m \times n$ image (Figure 2a) which must be first converted to double format using the built-in `im2double` function. The outputs, `im_dx` and `im_dy`, are the differentiated images along the x and y directions, respectively. To differentiate the image, use the 1D kernel [-1, 0, 1]. Note that you may need to pad zeros on the boundary of the input image to get the same size filtered images.

Submission Instructions: *Submit a MATLAB script, filter_image.m, that performs all the operations stated above. Generate the differential images by visualizing the magnitude of the filter response as shown*

in Figure 2b and 2c. You are required to include these images by embedding them in the report (do not submit the images separately). The MATLAB command `imagesc` *may be helpful in this regard.*

3 Gradient Computation



(a) (b) (c) (d) (e)

Figure 3: Visualization of the gradient magnitude (a), gradient orientation (b), and the gradients at every 3rd pixel (c-e). Note that the magnitudes have been rescaled for illustrative purposes.

Problem 3 (25 points)

Write a MATLAB function, `[grad_mag, grad_ang] = get_gradients(im_dx, im_dy)`, that computes the magnitude and angle of the gradient. The inputs, `im_dx` and `im_dy`, are the x and y differential images of size $m \times n$. The outputs, `grad_mag` and `grad_ang`, are the magnitude and orientation of the gradient images of size $m \times n$. Note that the range of the angle should be $\theta \in [0, \pi)$ and that the angle is unsigned (i.e. $\theta \equiv \theta + \pi$). By visualizing the gradients, you can see that 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.

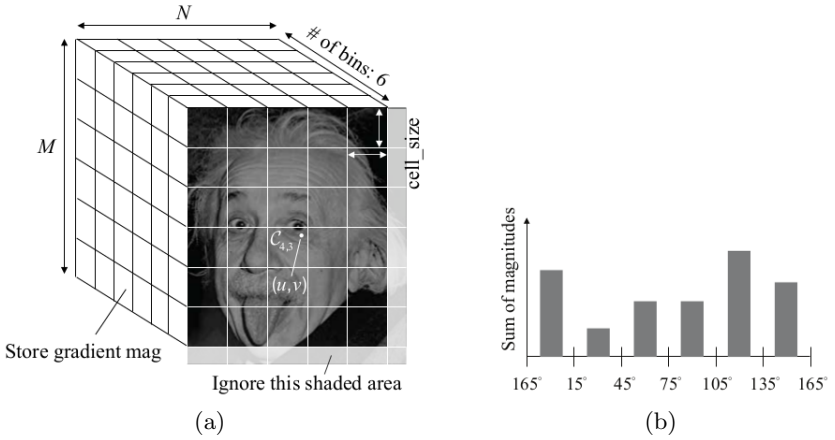
Submission Instructions: *Submit a MATLAB script, get_gradients.m, that performs all the operations stated above. Generate the gradient images by visualizing the magnitude and orientation of the gradients as shown in Figure 3a and 3b. You are required to include these images by embedding them in the report (do not submit the images separately). The MATLAB command* `imagesc` *may be helpful in this regard.*

4 Orientation Binning

Problem 4 (25 points)

Write a MATLAB function, `ori_histo = build_histogram(grad_mag, grad_ang, cell_size)`, that creates a histogram of oriented gradients for each cell. The first two inputs, `grad_mag` and `grad_ang`, are the magnitude and orientation of the $m \times n$ gradient images, respectively. The third input, `cell_size`, is a positive integer that indicates the size of each cell. A typical `cell_size` is 8. The output, `ori_histo`, is a 3D tensor of size $M \times N \times 6$ where M and N are the number of cells along y and x axes, respectively. Specifically, $M = \lfloor m/\texttt{cell\_size} \rfloor$ and $N = \lfloor n/\texttt{cell\_size} \rfloor$ where $\lfloor \cdot \rfloor$ is the floor function as shown in Figure 4a.

Given the magnitude and orientation of the gradients per pixel, we can build the histogram of oriented



(a) (b)

Figure 4: The histogram of oriented gradients (a) is built by placing the gradients in the corresponding bins (b).

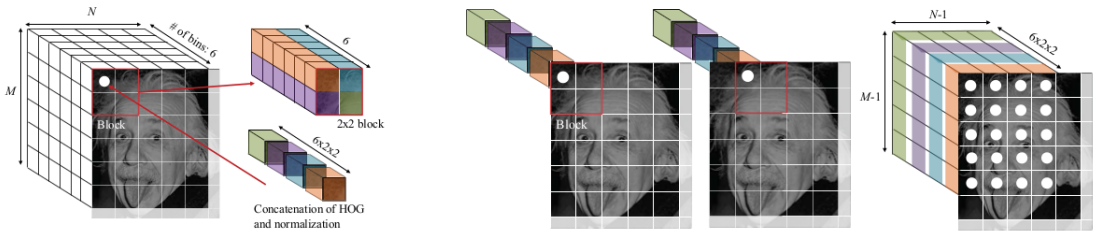
gradients for each cell

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

where $C_{i,j}$ is a set of x and y coordinates within the (i,j) cell and θ_k is the range of the angle in each bin. For example, $\theta_1 = [165°, 180°) \cup [0°, 15°)$, $\theta_2 = [15°, 45°)$, $\theta_3 = [45°, 75°)$, $\theta_4 = [75°, 105°)$, $\theta_5 = [105°, 135°)$, and $\theta_6 = [135°, 165°)$. Therefore, `ori_histo(i,j,:)` returns the histogram of the oriented gradients at each (i,j) cell as shown in Figure 4.

Submission Instructions: *Submit a MATLAB script, build_histogram.m, that performs all the operations stated above. Using* `ori_histo`*, generate images that visualize the HOGs per cell where the magnitude of the line proportional to the histogram is as shown in Figure 1. You are required to include these images by embedding them in the report (do not submit the images separately).*

5 Block Normalization



(a) 2×2 block descriptor. (b) Block descriptor overlap with a stride of 1.

Figure 5: The HOG descriptor is normalized to account for illumination and contrast, and to form a descriptor for each block. In (a), the HOG within block (1,1) is concatenated and normalized to form a long vector of size 24. In (b), this process is applied to the rest of the blocks with an overlap and stride of 1 to form the normalized HOG.

Problem 5 (25 points)

Write a MATLAB function, `ori_histo_normalized = get_block_descriptor(ori_histo, block_size)`, that builds the normalized histogram. The inputs, `ori_histo` and `block_size`, are the histogram of oriented gradients without normalization and the size of each block (i.e. the number of cells in each row/column), respectively. The output, `ori_histo_normalized`, is the normalized histogram of size $(M - (\texttt{block\_size} - 1)) \times (N - (\texttt{block\_size} - 1)) \times (6 \times \texttt{block\_size}^2)$.

To account for changes in illumination and contrast, the gradient strengths must be locally normalized. This requires grouping the adjacent cells together into larger, spatially connected blocks. To do this, given the histogram of oriented gradients we apply L_2 normalization as follows:

1. Build a descriptor of the first block by concatenating the HOG within that block using `block_size` $= 2$. A 2×2 block will contain $2 \times 2 \times 6$ entries that will be concatenated to form one long vector as shown in Figure 5a.
2. Normalize the descriptor,
$$\hat{h}_i = \frac{h_i}{\sqrt{\sum_i h_i^2 + e^2}},$$
where h_i is the ith element of the histogram, $\hat{h}_i$ is the normalized histogram, and e is a normalization constant that prevents division by zero (e.g. $e = 0.001$).
3. Assign the normalized histogram to `ori_histo_normalized(1,1)` (the location of the white dot in Figure 5a).
4. Move to the next block `ori_histo_normalized(1,2)` with a stride of 1 and reiterate steps 1-3, Figure 5b.

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

Submission Instructions: *Submit a MATLAB script, get_block_descriptor.m, that performs all the operations stated above.*

6 Application: Face Detection

Extra Credit (30 points)

Write a MATLAB function, `bounding_boxes = face_detector(im_target, im_template)`, that uses a HOG descriptor to perform face detection, Figure 6. The first input, `im_target`, is an image that contains multiple faces to be detected. The second input, `im_template`, is a template image that will be used to detect the faces. The output, `bounding_boxes`, is an $n \times 3$ array that describes the n detected bounding boxes.

Each row of `bounding_boxes` is an array of the form $[x_i, y_i, s_i]$ where (x_i, y_i) is the top-left corner coordinate of the ith bounding box, and s_i is the normalized cross-correlation (NCC) score between the bounding box patch and the template. The NCC score is calculated as

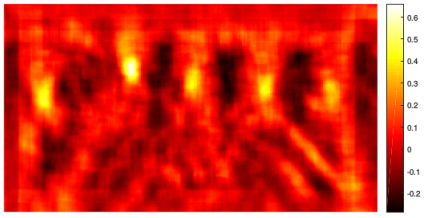
$$s = \frac{\bar{\mathbf{a}} \cdot \bar{\mathbf{b}}}{||\bar{\mathbf{a}}||||\bar{\mathbf{b}}||},$$

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

$$a_i = a_i - \tilde{\mathbf{a}},$$

(a) Template image.

(b) Target image.



(c) Response map.

(d) Thresholding.

(e) Non-maximum suppression.

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

where a_i is the ith element of $\mathbf{a}$, the HOG descriptor, and $\tilde{\mathbf{a}}$ is the mean of the HOG descriptor. To localize the faces, use thresholding and non-maximum suppression with a 50% Intersection over Union (IoU). The IoU is a measure of the overlap between two bounding boxes,

$$\text{IoU} = \frac{\text{area of overlap}}{\text{area of union}}.$$

Submission Instructions: *Submit a MATLAB script, face_detector.m, that performs all the operations stated above using the provided images 'template.png' and 'target.png'. In addition, run your face detector on a target image of your choice. Visualize the response maps along with the image results of thresholding and non-maximum suppression as shown in Figure 6. The MATLAB command* `rectangle` *may be used for drawing bounding boxes. You are required to include these images by embedding them in the report (do not submit the images separately).*

References

[1] N. Dalal and B. Triggs. "Histograms of Oriented Gradients for Human Detection". IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), vol. 1, pp. 886-893. 2005.

[2] P.F. Felzenszwalb, R.B. Girshick, D. McAllester and D. Ramanan. "Object Detection with Discriminatively Trained Part-Based Models". IEEE Transactions on Pattern Analysis and Machine Intelligence, 32(9), pp. 1627-1645. 2009.