RWTH AACHEN UNIVERSITY

Computer Vision
WS 2015/2016

Prof. Bastian Leibe<leibe@vision.rwth-aachen.de>
Stefan Breuers<breuers@vision.rwth-aachen.de>

Exercise 4: Histograms and Recognition

due before 2015-12-22

Important information regarding the exercises:

- In the archive for this exercise you will find the functions `apply.m` that should be used for displaying your results. You should also use it to test your implementation and see if the results make sense. Answers are to be submitted within `answers.m` Do **not** modify the apply files in any way.
- Please do **not** include the data files in your submission!
- Please submit your code solution as a zip/tar.gz file named `mn1_mn2_mn3.{zip/tar.gz}` with your **matriculation numbers** (`mn`).
- Please submit your solutions via the L^2P system.

Please note:

- The exercise is not mandatory.
- There will be no corrections. If you want to verify your solutions, use the provided `apply` functions.
- Nevertheless, we encourage you to work on the exercises and present your solutions in the exercise class. For this regard the above submission rules.

Question 1: Histogram Representations

a) The **hist** function of Matlab has two modes: When calledwithout specifying a return value, it displays a histogram of the vector given as an argument. When called with a specified return value, it returns the histogram instead of displaying it.

```
hist(image);                      % Displays the histogram of image
image_histogram = hist(image);  % Fills image_histogram with the
    histogram of image
```

Read an image and convert it to gray values. Reshape the 2D image array of size (N,M) to a 1D vector of size (N*M,1). Calculate the histogram of gray values using the **hist** function and display it using the **bar** function. Compare the histograms for different numbers of bins. Explain your observations.

Implement a function called `myhist` which takes a filename and a number of bins as an input and returns a 1D histogram vector of gray values. Use the **hist** function of Matlab for the histogram computation. Normalize the histogram such that its integral (sum) is equal to one.

```
function histogram = myhist(filename, bins)
```

Repeat the histogram computations using this new function. Does the result look the same?

b) Implement a new function `myhist2` which takes the same input as the previous one but returns a 3D histogram of RGB values.

```
function histogram = myhist2(filename, bins)
```

Optionally, remove the bins corresponding to the black color (background) before normalization.

Visualize the RGB histograms of the images `sunset.png` and `terrain.png` using the provided function `plot_color_histogram` and describe what you see. Which number of bins gives the best impression of the color distribution?

c) Implement a function `hist_dist_euclidean` which takes as an input two histograms `h1` and `h2` and returns the squared Euclidean distance between them:

$$\mathrm{dist}(h_1, h_2) = \sqrt{\sum_{i=1}^{D} (h_1(i) - h_2(i))^2}$$

```
function dist = hist_dist_euclidean(h1, h2)
```

Compute the distances between images `model/obj1__0.png` and `model/obj91__0.png`, `model/obj1__0.png` and `model/obj94__0.png`, which distance is smaller and why?

d) Implement 2 other distance functions you know from the lecture (except for Mahalanobis distance and Earth Mover's distance) and repeat the previous exercise.

e) (Bonus Question) Implement a new function `myhist3` which takes the same input as the previous one but returns a 2D histogram of `r`, `g` values, where $\mathbf{r} = \frac{R}{R+G+B}$ and $\mathbf{g} = \frac{G}{R+G+B}$.

```
function histogram = myhist3(filename, bins)
```

Apply this function to images of a red, a green and a blue object and visualize your results using **`imagesc`**:

```
h_vec = myhist3('model/obj4__0.png', 20);
histogram = reshape(h_vec, 20, 20);
imagesc(histogram);
colormap gray;
```

Do the histograms look correct?

f) (Bonus Question) Implement a new function `myhist4` which takes the same input as the previous one but returns a 2D histogram of `dx`, `dy` values, which are Gaussian derivatives of the image in x and y directions. Note that derivatives can have negative values. To correct that, add a constant value to the derivatives (e.g.,if your image contains values between 0 and 255, then the values of the derivatives will be in the range $[-255, 255]$.

```
function histogram = myhist4(filename, bins)
```

Visualize the gradient histograms of some images using **`imagesc`**. For getting a better visual impression, you should logarithmize the histograms:

```
h_vec = myhist4('model/obj4__0.png', 20);
histogram = reshape(h_vec, 20, 20);
imagesc(log(histogram));
colormap gray;
```

Do the histograms look correct?

Question 2: Recognition using Histograms

In this question, we will use the (provided) function `allhist` which takes as an input a file `model` with a list of image names and returns a 2D array of histograms of the images. The type of the histograms is specified by the argument `hist_type`. The histograms are the rows of the returned array.

```
function histograms = allhist(model, hist_type, bins)
```

a) Implement a function `histogram_query` which takes as an input a filename containing model names (use the supplied file `model.txt` for this) and a query image and returns the filename of the model image whose histogram had the minimal distance to the query image. The function should (a) compute an array of histograms for the model images and a histogram for the input image, (b) compute a vector of distances between the query histogram and each of the model histograms using a suitable distance function you implemented before, (c) find the minimum distance in the vector of distances, (d) use the index of the minimum distance to return the corresponding filename.

```
function best = histogram_query(model, query, bins)
```

Try your functions on some example images from the `query` folder and visualize the queries and the corresponding results.

b) Implement a function `eval_recognition_performance` which takes as input two filenames `model_list` and `query_list` containing the list of files in the model and query directories and determines the recognition rate, i.e. the ratio of correctly recognized objects and the error rate, i.e. the ratio of incorrectly recognized objects. Use the supplied files `model.txt` and `query.txt` for testing.

Compute two arrays of histograms; one for the model images and one for the query images. Then, implement a loop which computes a distance between each pair of histograms (model, query) and store the distances in a matrix. Determine the number of correct (incorrect) recognitions by counting the number of rows in the distance matrix in which the minimal element is (or is not) on the diagonal. Finally, divide these counts by the number of columns to normalize the score.

```
function [rec_rate, error_rate] = eval_recognition_performance(
    model_list, query_list, bins, hist_type)
```

c) (Bonus Question) Instead of finding minima in the distance array, find distances which are smaller than a chosen threshold, e.g. 0.3. Count the number of true and false detections. Divide the number of true detections by the number of columns to obtain the recognition rate (recall). Divide the number of false detections by the total number of distances below threshold to obtain the error rate (1-precision).

Modify the function from the previous exercise:

```
function [rec_rate, error_rate] =  eval_recognition_performance_thres(
    model_list, query_list, thres, bins, hist_type)
```

Compute the results for different thresholds `[0:0.01:0.4]`. Display a plot: true detections on Y axis and threshold on X axis. Display another plot: true detections on Y axis and negative detections on X axis. Explain what the two graphs show. What does a high or low threshold mean?

d) (Bonus Question) Use different methods for computing histograms and compare the resulting true detections and false detections. Repeat the exercise for different distance functions. Try to find a combination of histogram method and distance function which gives the best results. Write a short summary of your observations and your explanation for them.

Question 3: Sliding Window Detection

The goal of this exercise is to write a sliding window detector for cars on a single scale. A dataset containing training and test images will be provided[1]. As a feature vector we will use descriptors similar to HOG. The HOG descriptor collects histograms of oriented gradients in a number of cells across the image. The cells are ordered in a grid over the image and each cell has a constant size. In each cell a histogram is stored over the angles of the gradients. In our implementation we do not use the full circle ($[0, 2\pi)$) for the gradients. Instead we want to divide the half circle ($[0, \pi)$) into a given number of bins. To calculate this easily, use MATLAB's `mod` function which can also use floats. Pay attention to numerical problems here, since the angles can be almost zero. You have to handle this case in your implementation.

After calculating the HOG descriptor for each positive and negative training sample, the resulting descriptors are used to train a model with the help of Matlab's SVM implementation. In the testing step the trained model is evaluated on every possible window on the HOG representation of the test images. To clean up the initial hypotheses non maximum suppression and thresholding is used.

Write yout textual answers into `answers.m`.

a) Write a function to calculate a HOG-like descriptor. Since it is too complex to write the HOG descriptor itself in the scope of this exercise, we will restrict us to a simpler version. The function you should implement is

```
function hog = calculate_hoglike(img, cellsize, n_bins, sigma,
    offset_lr, offset_tb, interpolation)
```

where `img` is the grayscale image, `cellsize` is the size of each HOG-like bin in both dimensions, `n_bins` is the number of bins for the angles, `sigma` the scale of the Gaussian used to compute the image gradients and `offset_lr`, `offset_tb` margins in pixels to counteract border artifacts. The boolean flag `interpolation` is used in Subquestion 3b. The output should be a three dimensional matrix. The first two dimensions are the y and x indices of the HOG cell. The third dimension describes the bins of the HOG descriptor. When the dimensions of the images are not a multiple of the `cellsize`, discard the remaining pixels to the right and to the bottom of the image.

The function shall first calculate the gradient magnitude and direction for the whole image. Afterwards, it shall crop the image from left and right (`offset_lr`) and top and bottom (`offset_tb`). Each margin shall be applied on both sides. This is done to minimize the influence of boundary artifacts arising from gradient calculation.

Since the HOG descriptor is a cell based descriptor, we have to select for each pixel the cell (or with interpolation multiple) cells where the gradient is added. A simple version of this function is provided

```
function [cells_x, cells_y, weights] = get_cell(image_height,
    image_width, cellsize, x, y)
```

where `image_height`, `image_width` are the image dimensions, `cellsize` is the size of the HOG cell and `x` and `y` are the coordinates of the current pixel. It returns row vectors of the x and y coordinates and the weights for assigning the gradient to the corresponding cells. When the correct HOG cell and angle cell are selected, the product of weight and gradient magnitude is added to this cell.

Each cell has to be independently L_1 normalized to 1. Note that the original HOG descriptor uses a more elaborated two-stage normalization scheme.

Display the resulting descriptor calculated for some of the given images. For this you can use the function `render_hogimage(hog, output_size)`. What can you observe?

b) Extend the simple assignment scheme of pixels to HOG cells with a bilinear interpolation. The bilinear interpolation will distribute the gradient magnitude over multiple neighboring cells. This will result in a more robust descriptor.

Figure 1 explains the weighting scheme which will be applied. The areas of the colored rectangles are calculated. The weight for assigning the gradient to a given cell is the area of the opposing surface

[1]http://cogcomp.cs.illinois.edu/Data/Car/

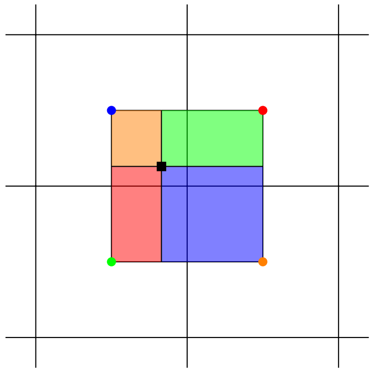

Figure 1: Bilinear Interpolation of HOG features.

divided by the sum of all areas. For example the weight for assigning the pixel gradient to the red cell is

$$w_{\text{red}} = \frac{A_{\text{red}}}{A_{\text{red}} + A_{\text{blue}} + A_{\text{green}} + A_{\text{orange}}}.$$

The formula for bilinear interpolation states

$$\begin{aligned}
h(x_1, y_1) &= h(x_1, y_1) + w \cdot \left(1 - \frac{x - x_1}{b_x}\right)\left(1 - \frac{y - y_1}{b_y}\right) \\
h(x_1, y_2) &= h(x_1, y_2) + w \cdot \left(1 - \frac{x - x_1}{b_x}\right)\left(\frac{y - y_1}{b_y}\right) \\
h(x_2, y_1) &= h(x_2, y_1) + w \cdot \left(\frac{x - x_1}{b_x}\right)\left(1 - \frac{y - y_1}{b_y}\right) \\
h(x_2, y_2) &= h(x_2, y_2) + w \cdot \left(\frac{x - x_1}{b_x}\right)\left(\frac{y - y_1}{b_y}\right)
\end{aligned}$$

Implement this method by creating the function

```
function [cells_x, cells_y, weights] = get_cell_weights(image_height,
    image_width, cellsize, x, y)
```

with the same signature as described in Exercise 3a. Modify the function `calculate_hoglike` to make use of this improvement when the flag `interpolation` is set.

c) Since the SVM has created a model we can now use this for evaluating our detector on test images. For this you should create a function

```
function scores = object_hypothesis(svm, hog_img, ny_bins, nx_bins)
```

where `svm` is a model instance of the SVM, `hog_img` is a grid of HOG cells from a test image and `ny_bins`, `nx_bins` describe the size of the window (the number of cells in y or in x direction respectively). The function should output a matrix of the detection scores. Since the window spans several cells the result matrix `scores` is smaller than the input `hog_img`.

For each possible window the feature vector is computed and tested with the SVM. A valid window lies completely in the area of the test image. To speed up the computation the vectors should be written to a matrix where each row is a feature vector. You can use the provided function `calculate_svm_score(svm, windows)` to compute the score for each window. To only keep the local maxima, non maximum suppression is used. For this, use the provided function `nms`.

To retrieve the detected objects implement the function

```
function [y,x, score] = detect_objects(hypotheses, threshold)
```

The inputs are the SVM scores `hypotheses` and a `threshold`. The function should return row vectors of y and x coordinates and the score at these points where the score is larger than the given `threshold`.

Evaluate the test images. Can you see where this detection fails?

d) (Bonus Question) Multiple detections sometimes overlap. To counteract this behavior, modify the non maximum suppression method to incorporate the following method applied after the previous completed non maximum suppression. First, all possible detections are sorted in descending order according to their score and initially no detection is accepted. If the current detection does not overlap significantly with an accepted one, add this one to the result set. To decide whether the overlap is significant, use the intersection-over-union criterion:

$$\mathrm{iou}(\mathrm{bbox}_1, \mathrm{bbox}_2) = \frac{\mathrm{bbox}_1 \cap \mathrm{bbox}_2}{\mathrm{bbox}_1 \cup \mathrm{bbox}_2}.$$

If the iou of two detections (here described by bounding boxes) is larger than 0.5, the detection is discarded.

Incorporate this method into the function

```
function result = nms_box(score, ny_bins, nx_bins)
```

e) (Bonus Question) To get an even more robust HOG descriptor use trilinear interpolation instead of bilinear interpolation. Trilinear interpolation also takes into account that the gradient is estimated with noise. Therefore, the gradient product of weight and gradient magnitude is distributed over neighboring cells in the descriptor. As depicted in Figure 2, the gradient is not added direcly to

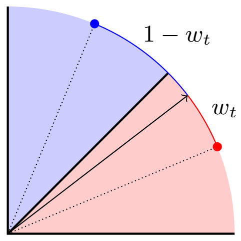

Figure 2: Trilinear interpolation.

the red cell. Instead it is added to both the blue and the red cell with a weight of $1 - w_t$ and w_t respectively. The statement $0 \leq w_t \leq 1$ holds. It might happen that the gradient direction falls directly on the center of a cell. This might create ambiguities in your implementation when selecting the neighboring cell.

Implement a function

```
function assignments = trilinear_interpolation(n_bins,
    gradient_direction)
```

which returns the `assignment` vector which contains the weight for each cell for one HOG bin. The function receives the number of bins `n_bins` and the `gradient_direction`. Incorporate this new function into the function `calculate_hoglike` when the flag `interpolation` is set. How do the detections change?

Please turn in your solutions including all relevant files before 2015-12-22!