*Total # of points = 100.*

**Project description.** In this project, you will implement a program that uses *HOG* (*Histograms of Oriented Gradients*) and *LBP* (*Local Binary Pattern*)features to detect human in images. First, you will use the HOGfeature only to detect humans. Next, you will combine the HOG feature with the LBPfeature to form an augmented feature (HOG-LBP) to detect human. A *Two-Layer Perceptron* (feedforward neural network) will be used to classify the input feature vector into *human* or *no-human.*

**Conversion to grayscale:** The inputs to your program are color sub-images cut out from larger images. First, convert the color images into grayscale using the formula where *R, G* and *B* are the pixel values from the red, green and blue channels of the color image, respectively, and *Round* is the round off operator.

**Gradient operator:** Use the **Sobel’s operator** for the computation of horizontal and vertical gradients. Use formula to compute gradient magnitude, where are the horizontal and vertical gradients. Normalize and round off the results to integers within the range [0, 255]. Next, compute the gradient angle (with respect to the positive *x* axis that points to the right.) For image locations where the templates go outside of the borders of the image, assign a value of 0 to both gradient magnitude and gradient angle*.* Also, if both are 0, assign a value of 0 to both gradient magnitude and gradient angle.

**HOG feature:** Refer to the lecture slides for the computation of the HOG feature. Use the unsigned representation and quantize the gradient angle into one of the 9 bins as shown in the table below. If the gradient angle is within the range [170, 350), simply subtract by 180 first. Use the following parameter values in your implementation: *cell size* = 8 x 8 pixels, *block size* = 16 x 16 pixels (or 2 x 2 cells), *block overlap* or *step size* = 8 pixels (or 1 cell.) Use *L2* norm for block normalization. Leave the histogram and final feature values as floating point numbers. Do not round off to integers.

|  |  |  |
| --- | --- | --- |
| **Histogram Bins** | | |
| Bin # | Angle in degrees | Bin center |
| 1 | [-10,10) | 0 |
| 2 | [10,30) | 20 |
| 3 | [30,50) | 40 |
| 4 | [50,70) | 60 |
| 5 | [70,90) | 80 |
| 6 | [90,110) | 100 |
| 7 | [110,130) | 120 |
| 8 | [130,150) | 140 |
| 9 | [150,170) | 160 |

**LBP feature:** For the computation of the LBP feature, first divide the input image into non-overlapping blocks of size Next, compute LBPpatterns (refer to lecture slides) at each pixel location inside the blocks and convert the 8-bit patterns into decimals within the range [0, 255]. Then, form a histogram of the LBP patterns for each block. To reduce the dimension of the histogram, we create separate bins for uniform patterns and a single bin for all non-uniform patterns. An 8-bit LBP pattern is called uniform if the binary pattern contains at most two 0-1 or 1-0 transitions if we go around the pattern in circle. For example, 00010000 (2 transitions) is a uniform pattern, but 01010100 (6 transitions) is not. By putting all non-uniform patterns into a single bin, the dimension of the histogram is reduced from 256 to 59. The 58 uniform binary patterns correspond to the integers 0, 1, 2, 3, 4, 6, 7, 8, 12, 14, 15, 16, 24, 28, 30, 31, 32, 48, 56, 60, 62, 63, 64, 96, 112, 120, 124, 126, 127, 128, 129, 131, 135, 143, 159, 191, 192, 193, 195, 199, 207, 223, 224, 225, 227, 231, 239, 240, 241, 243, 247, 248, 249, 251, 252, 253, 254 and 255, and all other integers belong to non-uniform patterns. Let the 1st to 58th bins of your histogram be assigned to the uniform patterns according to the order above, and the 59th bin be assigned to non-uniform patterns. For pixels in the first and last rows, and first and last columns of the image, we cannot compute their LBP patterns since some of their 8-neigbors are outside of the borders of the image. Simply assign a LBP value of 5 at these pixel locations and they will be assigned to the 59th bin of the histogram for non-uniform patterns. Finally, concatenate the histograms from all blocks (in left to right, then top to bottom order) to form a single feature vector.

**HOG-LBP feature:** To form the combined HOG-LBP feature, simply concatenate the HOG and LBP feature vectors together to form a long vector.

**Two-layer perceptron**: Implement a fully-connected two-layer perceptron with an input layer of size *N*, with *N* being the size of the input feature vector, a hidden layer of size *H* and an output layer of size 1. Let (two experiments) and report the training and classification results for each. (Optional: you can try other hidden layer sizes and report the results if you get better results than the two above.) Use the *ReLU* activation function for neurons in the hidden layer and the *Sigmoid* function for the output neuron. The Sigmoid function will ensure that the output is within the range [0,1], which can be interpreted as the probability of having detected *human* in the image. Use the weight updating rules we covered in lecture for the training of the two-layer perceptron. Use random initialization to initialize the weights of the perceptron. Assign an output label of 1.0 for training images containing human and 0.0 for training images with no human. You can experiment with and decide on the learning rate to use (can try 0.1 first.) After each epoch of training, compute the *average error* from the errors of individual training samples. The error for an individual training sample , with the correct output equals 1.0 for positive samples and 0.0 for negative samples. You can stop training when the change in average error between consecutive epochs is less than some threshold (e.g., 0.1) (0.01,0.001might better) or when the number of epochs is more than some maximum (e.g., 1000.) *Mini-batch/online training/batch decide by your own.* After training, you can use the perceptron to classify the test images. Use the following rules for classification:

|  |  |
| --- | --- |
| **Perceptron output** | **Classification** |
|  | human |
|  | borderline |
|  | no-human |

**Training and test images**: A set of 20 training images and a set of 10 test images in .*bmp* format will be provided. The training set contains 10 positive (human) and 10 negative (no human) samples and the test set contains 5 positive and 5 negative samples. All images are of size 160 (height) X 96 (width). You can download the images from:

https://drive.google.com/drive/folders/1Lk7FLJ4fIpBZ708pOwEI-RWaGa-I35ky?usp=sharing

To access, you need to log on Google Drive with your NYU NetID.

**Experiments:** You need to perform experiments with hidden layer sizes of 200 and 400 in the perceptron, and for each hidden layer size, use the HOG only feature and then the combined HOG-LBP feature (a total of four experiments.) **(a)** **HOG only feature.** Given the image size of and the parameters given above for HOG computation, you should have 20 X 12 cells and 19 X 11 blocks. The size of your feature vector (and the size of the input layer of your perceptron) is therefore 7,524. (b) **Combined HOG-LBP feature.** With the parameters given above for the LBP feature, there are blocks in the input image and the size of the LBP feature is The combined HOG-LBP feature therefore has size

**Implementation:** You need to write program code to implement the *HOG* and *LBP* features, and the *two-layer* *perceptron*. You can use Python, C++/C, Java or Matlab to implement your program. If you would like to use a different language, send me an email first. You are not allowed to use any built-in library function for any of the tasks that you are required to implement, including the Sobel’s operator, computation of the HOG and LBP features, and the two-layer perceptron. The only library functions you are allowed to use are those for the reading and writing of image files, matrix and vector arithmetic, and certain other commonly used mathematical functions.

**Hand-in:** Hand in the following on NYU Classes by the due date. Please submit as separate files, do not ZIP.

* Your source code file. Put comments in your source code to make it easier for someone else to read your program. Points will be taken off if you do not have this.
* An ASCII (.txt) file containing the HOG feature values for the image *crop001034b* and a separate ASCII (.txt) file containing the LBP feature values for the same image*.*  The feature values should be separated by line breaks. You should have 7,524 lines in the HOG file and lines in the LBP file.
* A PDF report that contains the following:
  + File names for your source code and the HOG and LBP feature files for image *crop001034b*.
  + Instruction on how to run your program, and instruction on how to compile your program if your program requires compilation.
  + Method you used to initialize the weight values of your perceptron (e.g. random initialization with values within range [0.0, 1.0].)
  + Criteria you used to stop training (e.g., when change in *average error* between consecutive epochs is less than 0.1 or when number of epochs reaches 1000.)
  + The number of iterations (or epochs) required to train your perceptron. Report for each of the four experiments: hidden layer sizes of 200 and 400 -- HOG only and combined HOG-LBP.
  + For hidden layer sizes 200 and 400, create separate tables (see below) that contain the output values of the output neuron and the classification results (human, borderline or no-human) for HOG feature only and for the combined HOG-LBP feature. Use the rules above (in **Two-layer perceptron** section) for classification. Report results for all 10 test images in the table. Also, compute and report the *average error* for the 10 test images. The error for a test sample is computed as .
  + Any other comments you may have on your program, training and testing of the perceptron and your results.
  + Normalized gradient magnitude images for the 10 test images (Copy-and-paste from output image files.)
  + The source code of your program (Copy-and-paste from source code file. This is in addition to the source code file that you need to hand in.)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Image** | **Correct Class** | **HOG only** | | **HOG-LBP** | |
|  |  | **Output** | **Classification** | **Output** | **Classification** |
| crop001034b | Human |  |  |  |  |
| crop001070a | Human |  |  |  |  |
| crop001278a | Human |  |  |  |  |
| crop001500b | Human |  |  |  |  |
| person\_and\_bike\_151a | Human |  |  |  |  |
| 00000003a\_cut | No-human |  |  |  |  |
| 00000090a\_cut | No-human |  |  |  |  |
| 00000118a\_cut | No-human |  |  |  |  |
| no\_person\_no\_bike\_258\_cut | No-human |  |  |  |  |
| no\_person\_no\_bike\_264\_cut | No-human |  |  |  |  |

Output [0,1.0]

Cannot use library of implementing preceptron

**Hints:**

* When computing HOG, since we subtract 180 from the gradient angle if the angle is larger than 180, we can consider each bin to have two bin centers; for example, for bin 9, the two bin centers are 160 and 160+180 = 340 = -20 degrees. So, if the gradient angle of a pixel falls between bin centers  - 20  and 0 degrees, the vote should be split between bins 9 and 1.
* Since HOG is normalized within blocks, you should also normalize LBP within blocks so that HOG and LBP are similar in scale.  This will give better results when training  the MLP. Within each block, you can normalize by dividing each  histogram bin by 256, which is the total pixel count in the block (16 x 16 = 256.)  In the output feature vector file, you can write the  normalized values.
* In the MLP, every neuron in the hidden layer and the neuron in the output layer has a bias input of -1 and a bias weight for the link with the bias input. You would train the bias weight together with the weights of other input links.
* To get better results when training the MLP, you should perform random initialization of the weights.