Project-4

CIS 666 Artificial Intelligence

Histogram Oriented Gradients and pedestrian detection using SVM Classifier in python

Fathima Syeda

Student ID: 2790024

Abstract

This project implements a HOG Feature extractor and makes use of the SVM classifier for pedestrian detection in the NICTA Pedestrian Dataset. The training set contains 1000 positives samples (images contain pedestrians) and 2000 negatives samples (images do not contain pedestrians). The testing set includes 500 positive samples and 500 negative samples. In the implementation of HOG, the gradient(magnitude, orientation) is found for the image. Then using a sliding window of 16x16 we get 105 overlapping blocks with 50 percent. For each of these blocks four 8x8 cells are taken and their histograms are calculated and concatenated . Finally, the concatenated Feature vector for all the blocks is obtained. This way the HOG descriptors for both the positive, negative samples of both the training and test data are found . The SVM classifier is trained on the training data with different kernels(linear, rbf, polynomial) and the test data is tested on the classification model to depict the accuracy, false positive rate, and the miss rate.

Contents

1. Introduction:

HOG(Histogram of Oriented Gradients) is a kind of feature descriptor . A feature descriptor is a representation of an image or an image patch that simplifies the image by extracting useful information and throwing away extraneous information. The HOG descriptor gives a vector of 3780 from an input image of 64x128x3. A feature vector is useful in object detection and image recognition as it ignores the extraneous data and only gives the useful data. The feature vector gotten from HOG can be given to the Support Vector Machine Classifier (SVC) for training of model that will be used for classification.

In the HOG feature descriptor, the distribution (histograms) of directions of gradients (oriented gradients) are used as features. Gradients (x and y derivatives) of an image are useful because the magnitude of gradients is large around edges and corners (regions of abrupt intensity changes) as the edges and corners pack in a lot more information about object shape than flat regions.

The gradients are calculated by multiplying the image with the following kernels to get Gx, Gy.

-1	0	1

-1
0
1

The following images depict the gradients in x, y directions and their magnitudes when visualised.

Left : Absolute value of x-gradient. Centre : Absolute value of y-gradient. Right : Magnitude of gradient.

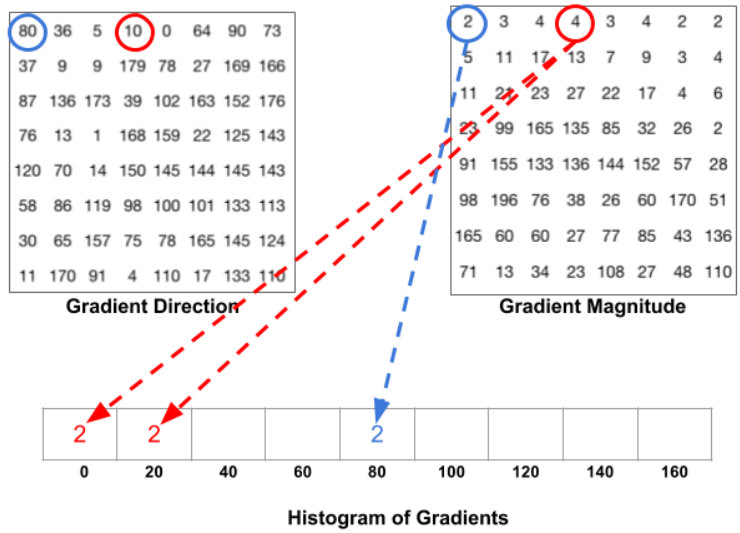
The horizontal lines are fired along the y-gradient and the horizontal lines along the x-gradient. The magnitude of gradient fires wherever there is a sharp change in intensity. None of them fire when the region is smooth. The gradient image removes a lot of non-essential information (e.g. constant coloured background), but highlights outlines.

At every pixel, the gradient has a magnitude and a direction. For colour images, the gradients of the three channels are evaluated (as shown in the figure above). The magnitude of gradient at a pixel is

the maximum of the magnitude of gradients of the three channels, and the angle is the angle corresponding to the maximum gradient.

Magnitude G = (G^2_x+ G^2_y)$^{1/2}$ and the Orientation is Θ = Tan^{-1}(G_y / G_x)

The input image is taken and divided into 16x16 overlapping blocks and from each block four 8x8 cells are taken and the histogram is formed for each of them. If the orientation is "**signed**" its values will range from 0-360 and if it is "**unsigned**" the values would range from 0-180. The bin width of the histogram is given as $\pi/\mathbf{B}$ where B is the number of bins.



The gradient direction of each pixel is taken, and the gradient magnitude is used as the weighted vote to sort it and place it into a bin. In the above picture the pixel with gradient direction 80 belongs to the Bin=80 so its gradient magnitude is taken as the weighted vote and put into the Histogram. Similarly, the gradient orientation 10 is in between Bins=0 and bins=20 , so it's gradient magnitude 4 will be split into two and put in both the bins 0 and 20. This way the histogram for each cell is found.

These histograms (9x1) of 8x8 cells in each of the 16x16 blocks (105 blocks) are of the dimensions 36x1. They are normalized so that they are not affected by the lighting variations by means of

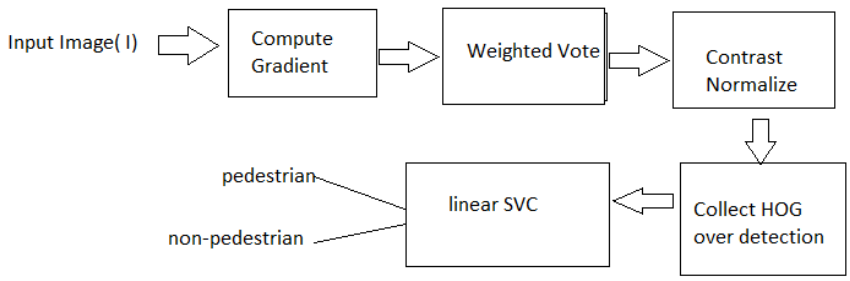
L2norm = $\dfrac{v}{\sqrt{\sum \mathbf{abs}(v^2) + \mathbf{E}}}$

After normalizing each vector in the 16x16 block we concatenate them together to get the Final HOG Feature Descriptor of 1x3780 .

This feature vector of all the images is then fed to the SVM Classifier to train the model to distinguish an image with a pedestrian in it from one with no pedestrian in it.

Mallick ,Satya. "Histogram of Oriented Gradients". Learn OpenCV , 6 Dec. 2016,
https://www.learnopencv.com/histogram-of-oriented-gradients/

2. Steps involved in HOG Feature Extraction



The above figure shows the steps in detection of pedestrian or no pedestrian in the image

1. Read the input image and resize it to 64x128.
2. Convolve the image by multiplying it with kernel h_x= [-1 0 1] to get the gradient in x direction G_x and kernel h_x= $[-1\ 0\ 1]^T$ to get the gradient in x direction G_y.
3. Compute the gradient magnitude and orientation as Magnitude G = $(G^2_x + G^2_y)^{1/2}$ and the Orientation is $\Theta = \text{Tan}^{-1}(G_y / G_x)$
4. Get 105 (7*15) overlapping blocks (50 percent overlap) of size 16x16. In each of these 16x16 blocks get four 8x8 cells.
5. For each cell compute histogram of gradient By:
   (1) Divide edge direction $\boldsymbol{\Theta}$ into B bins with spacing $\boldsymbol{\pi/B}$
   (2) Accommodate the gradient magnitude of each pixel within a cell C_i into a bin with respective to its gradient direction to form histogram h_{Ci}.
   (3) Concatenate the histogram of cells in each block to form single vector h_{bj} =[$h_{C1,}$ $h_{C2,\ldots}$]
6. Normalize this block long vector H_{bj} using L2norm
   As $h'_{bj} = \frac{\boldsymbol{v}}{\sqrt{\sum \mathbf{abs}(\boldsymbol{v^2}) + \mathbf{E}}}$
7. Concatenate these normalized histograms across all the overlapping blocks to form the HOG descriptor.
8. Get the HOG descriptors for all the training image samples both positive and negative samples and send them to the SVC for training to built a model.
9. Get the HOG descriptors for all the test image samples both positive and negative samples and test them to get the accuracy, false positive rate, miss rate.
10. Repeat steps 8, 9 with different SVC kernels= linear, poly, rbf etc.

3. Program Implementation

The program consists of 4 functions to implement the single layer perceptron algorithm to correctly identify handwritten images into 10 digits from 0-9. The following are the functions:

- sliding_window(image, orientation, step Size, window Size)
- HOG_FeatureExtractor(I)

- getHOGDescriptors(Positive Images, Negative Images)
- SVM_Classification(TrainingData,TrainLabels,TestData,TestLabels,k='linear')

3.1 sliding_window(image, orientation, step Size, window Size)

This function returns blocks from the given image gradient and its orientation based on the given window size and decides the overlap of these blocks using the given step size.

3.2 HOG_FeatureExtractor(I)

This function finds the HOG of the given image by dividing into 105(7*15) overlapping blocks by calling the sliding_window() function. It then gets four 8x8 cells from each 16x16 block and finds its histogram and concatenates them to get 36x1 vector. It then normalizes these vectors using the numpy.l2norm function and concatenates them all to get a final HOG vector 3780x1.

3.3 getHOGDescriptors(Positive Images, Negative Images)

This function gets the hog descriptors for all the images in the given positive and negative samples. The hog descriptors of both the positive and negative samples are concatenated together. The labels array is found for the positive and negative samples by putting 1 for positive sample and 0 for negative sample. The hog descriptors of all the images and their corresponding labels list are returned.

3.4 SVM_Classification(TrainingData,TrainLabels,TestData,TestLabels,k='linear')

This function takes the training data and the testing data along with their corresponding labels. The training and the testing data are the HOG descriptors of all the images. It also takes in an optional parameter k for the kernel of the SVC classifier. The data is trained and tested using in-built functions. Then the accuracy, false positive rate and ,miss rate is found and printed.

accuracy $= \frac{true\ positive + true\ negative}{true\ positive + true\ negative + false\ positive + false\ negative}$

$false\ positive$ rate= $\frac{false\ positive}{true\ negative + false\ positive}$

Miss-rate= $\frac{false\ negative}{true\ positive + false\ negative}$

4. Testing

The positive and negative samples are shuffled instead of simply concatenating them and fed to the SVC for training. The result is as follows .

- When the HOGs of all the training data samples are trained on an SVM classifier with **kernel=linear** the results are as:
    - Accuracy:0.502
    - False positive rate:0.36
    - Miss Rate:0.636
- When the HOGs of all the training data samples are trained on an SVM classifier with **kernel=rbf** the results are as:
    - Accuracy:0.5
    - False positive rate:0.0
    - Miss Rate:1.0

- When the HOGs of all the training data samples are trained on an SVM classifier with **kernel=polynomial** the results are as:
    - Accuracy:0.499
    - False positive rate:0.058
    - Miss Rate:0.944

```
------------
training the SVM classifier with different kernels
For SVM classified with kernel=linear
Accuracy:0.474
False positive rate:0.348
Miss Rate:0.704
------------
For SVM classified with kernel=rbf
Accuracy:0.5
False positive rate:0.0
Miss Rate:1.0
------------
For SVM classified with kernel=poly
Accuracy:0.502
False positive rate:0.046
Miss Rate:0.95
```

We notice that when the positive and negative samples are shuffled instead of simply concatenating to a list before sending it for training, the accuracy reduces by almost half.

5. Obtained Results

The dataset used in this project is a NICTA pedestrian dataset which has 2000 negative and 1000 positive training sample images and 500 positive and 1000 negative test sample images. Each image is taken, and its HOG is found.

For each image of size 64x128 we get 105 blocks of 16x16 (with 50 % overlap). For each of these 16x16 blocks we get four 8x8 cells and get the histogram of 9x1 . Thus, each block has a vector of 36x1 . The final HOG is the concatenated vector of all the 105 blocks which is a vector of 3780x1.

The positive and negative samples are simply concatenated and given to the SVM for training.

1) When the HOGs of all the training data samples are trained on an SVM classifier with **kernel=linear** the results are as:
   Accuracy: 0.951
   False positive rate: 0.026
   Miss Rate: 0.072
2) When the HOGs of all the training data samples are trained on an SVM classifier with **kernel=rbf** the results are as:

   Accuracy: 0.95

   False positive rate: 0.014

   Miss Rate: 0.086

3) When the HOGs of all the training data samples are trained on an SVM classifier with **kernel=polynomial** the results are as:

Accuracy: 0.955

False positive rate: 0.012

Miss Rate: 0.078

```
------------
training the SVM classifier with different kernels
For SVM classified with kernel=linear
Accuracy:0.951
False positive rate:0.026
Miss Rate:0.072
------------
For SVM classified with kernel=rbf
Accuracy:0.95
False positive rate:0.014
Miss Rate:0.086
------------
For SVM classified with kernel=poly
Accuracy:0.955
False positive rate:0.012
Miss Rate:0.078
```

6. Conclusion

- The HOG descriptor is much more accurate at extracting features than other descriptors like the (Haar cascade) as it also takes the orientation into account .
- For the given NICTA data set the SVM classifies most accurately with linear kernel followed by the polynomial kernel and then the rbf kernel.
- The rbf kernel takes the most time in training followed by polynomial and the linear SVC takes the least time
- When the positive and negative samples are shuffled instead of simply concatenating to a list before sending it for training, the accuracy reduces by almost half.

7. References

- Mallick, Satya. “Histogram of Oriented Gradients”. Learn OpenCV , 6 Dec. 2016, https://www.learnopencv.com/histogram-of-oriented-gradients/
- Rosebrock, Adrian.” Histogram of Oriented Gradients and Object Detection”. Pyimagesearch, 14 Nov. 2014, https://www.pyimagesearch.com/2014/11/10/histogram-oriented-gradients-object-detection/
- Class Notes