1 - IMPLEMENTATION

The goal of this project is to implement a variant of Histograms of Oriented Gradients (HOG) object detector by Dalal&Trigggs.[1] *HOG can be computed efficiently using integral gradient images, which are analogous to the standard integral image (Viola&Jones, Section 2).*[2]

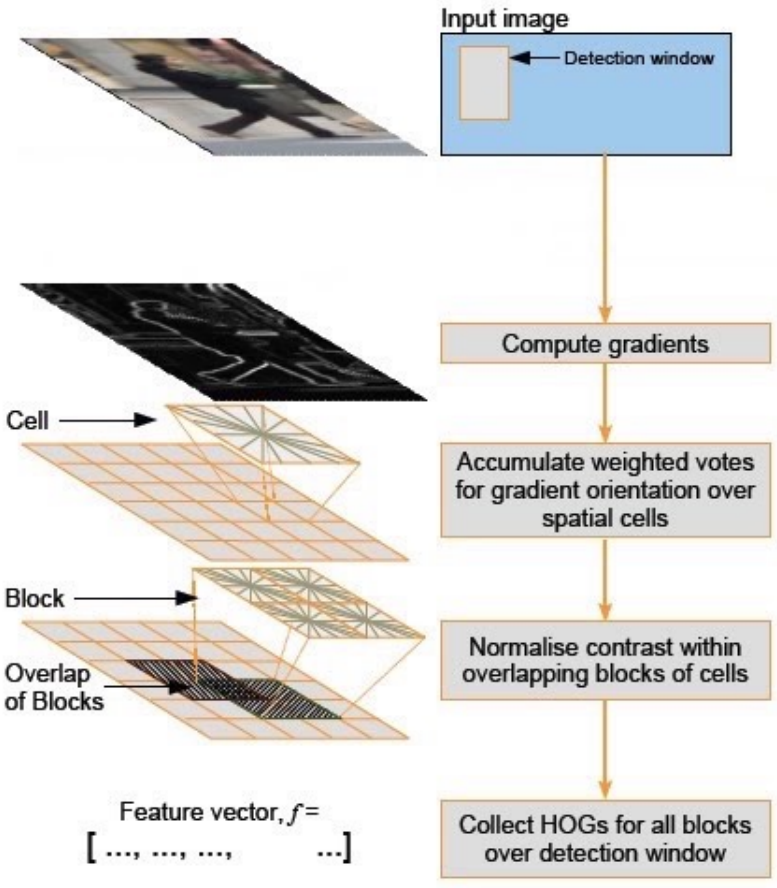
Given an image we want to compute the change (gradient) on both x and y axis and we are going to operate on gradient field, particularly considering orientation changes. Following the algorithm implemented includes quantisation of orientations for each patch into pre-defined buckets(bins). This quantisation of orientations may use a weight scheme. In the implementation, orientations are weighted by magnitudes of the gradients. While we scanning the particular image of interest, for every patch(block) histograms of weighted orientations are generated (HOG descriptors) and compared with template patch. Comparison is based on euclidian distance so histograms in HOG descriptor are normalised by their L2-norms. Generating HOG descriptors is computational expensive, hence algorithm benefits from integral image concept from Viola&Jones[2].

HOG

HOG Feature extractions can be summarised as below;

- Compute centered horizontal and vertical gradients with no smoothing.
- Compute gradient orientation and magnitudes. (For color image, one can pick the color channel with the highest gradient magnitude for each pixel location)

The overview of HOG Feature extraction[4]

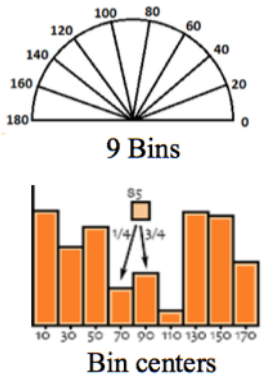
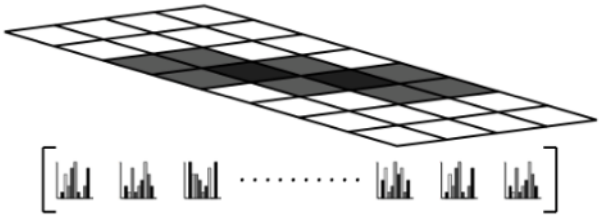


The specific steps for the HOG algorithm implemented are for object detection;

- Image is divided to the patches of arbitrary size and scanned with arbitrary **STEPSIZE**=4.
- Patches are divided to sub-patches and gradients are calculated. Then the gradient orientations quantised into **QNUM**=9 bins (*Tri-linear Interpolation*). The votes of the histogram quantisation are gradient magnitudes (how strong is that gradient direction using the gradient magnitude).
- *HOG Descriptors*: histograms are concatenated to generate HOG descriptors for each patch.

Tri-linear Interpolation;
For each block of particular size are divided to subpatches of size (nx,ny; n=4) and n^2 gradient is calculated on sub-blocks. Then calculations are interpolated (into 9 bins 0-180 deg.) based on orientation values as votes between neighbouring bin centers weighted by magnitude values. Histograms for each block in the HOG descriptor are normalised by their L2-norms.

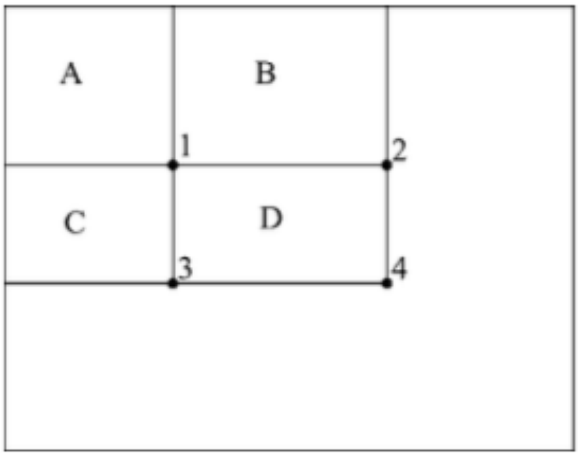
HOG Descriptors (feature vector);
For each path histograms are concatenated to generate a 1D-vector of weighted orientation histograms. This vector implies the HOG descriptor of the particular patch and compared by pythagorean metric to template patch. Looking at the histogram one can say what's the dominant direction in the particular patch.

Tri-linear Interpolation[4]	HOG Descriptors (feature vector)[4]
	

INTEGRAL IMAGE (Gradient histogram integrals)

Feature extraction for HOG is expensive in real-time situations. What we are going to do is benefit from the basic idea that, every time we calculate histograms for gradients that doesn't look computational efficient on block-basis. Viola&Jones come up with an idea called integral image (Viola&Jones, Section 2).[2] Based on this idea, calculation does not have to be made in patch-basis but instead we sum the corners of the patch cumulatively as; in an integral image the value at pixel(x,y) is the sum of the pixels above and to the left of (x,y).

The advantage of calculating the integral image is that allows for the calculation of sum of all pixels inside any given rectangle using only four values at the corner of the rectangle. An integral histogram representation can be used for fast calculation of histograms of oriented gradients over arbitrary rectangular regions of the image. The idea of an integral histogram is analogous to that of an integral image, used by Viola&Jones for fast calculation of HAAR features for face detection;

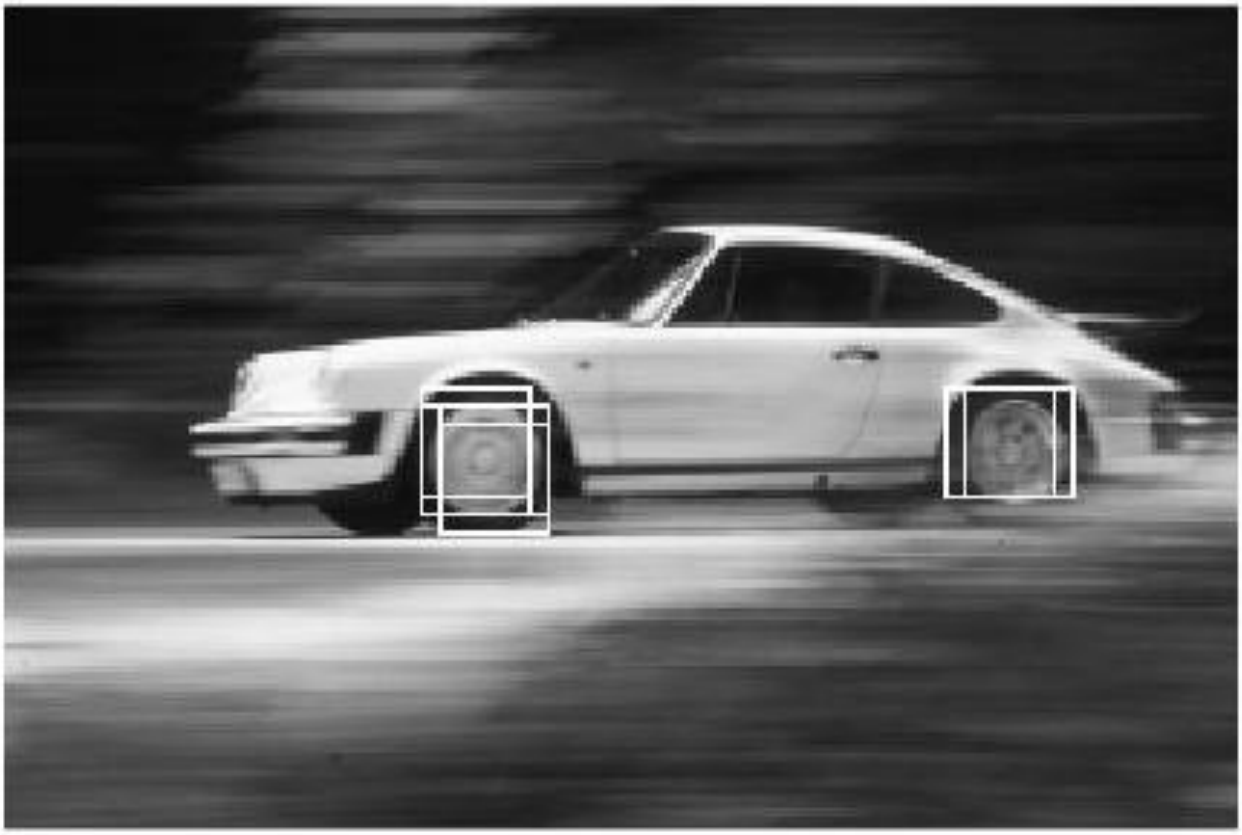
Integral histogram image calculation	
$$ih(x,y,b)=\sum_{x'\le x}\sum_{y'\le y}h(x',y',b)$$  *Figure 3.* The sum of the pixels within rectangle D can be computed with four array references. The value of the integral image at location 1 is the sum of the pixels in rectangle A. The value at location 2 is $A+B$, at location 3 is $A+C$, and at location 4 is $A+B+C+D$. The sum within D can be computed as $4+1-(2+3)$. *(Viola&Jones, Section 2, fig.3)* [2]	Where b represents the bin number of the histogram. Calculation of HOG over an arbitrary rectangle in the image requires just 4Xbins number of array references.

"Integral images facilitate summation of pixels and can be performed in constant time, regardless of the neighbourhood size."[3] Integral gradient image computed above enables to obtain a histogram of gradient orientations for any image rectangle efficiently in constant time.

2 - ANALYSIS

A test case of the algorithm for image ‘car1.png’ and ‘car2.png’ is given below;

Output of the test case for “car1.png” and “car2.png” (STEPSIZE=4, HOGTHRS=11)



The algorithm implemented for object detection/localization uses two parameters assigned by the user, **STEPSIZE** and **HOGTHRS**. The implementation gives fairly good results as seen above. #6 patches found as implied with white frames. The following parameters are assigned to algorithm;

```
STEPSIZE = 4            # Frame scanning step size
HOGTHRS  = 11           # HOG distance threshold
```

Even the with the benefit integral image technique, the algorithm is computational expensive and takes more than few seconds on 273x182 pixel grayscale image if one want to scan all possible patch locations on an image. Hence, scanning step size can be adjustable by the parameter STEPSIZE. The following min. matching HOG distances are calculated on the 273x182 pixel grayscale image 'car2.png' with different step sizes;

```
min. distance [STEPSIZE:16]: 11.54
min. distance [STEPSIZE:8]: 9.49
min. distance [STEPSIZE:4]: 9.49
min. distance [STEPSIZE:2]: 9.29
min. distance [STEPSIZE:1]: 8.75
```

Pythagorean metric is used for calculating HOG distances. Since gradient histograms are normalised by their L2-norms, comparing euclidian distances make sense. Since normalisation by L2-norms, histograms doesn't compensate to probability distributions. Also we see that the square of the distance between the QNUM (bin count) points gets an additive quadratic contribution from the separation in the new dimension of the QNUM points.

About invariance, we can easy say HOG descriptors, hence the algorithm is spatial-invariant as after we decide upon template patch, we scan all across the image in interest. Though we can state the same intuition about scale and rotation invariance. The example run doesn't say much about the rotation-invariance since the template patch we choose circular symmetric. HOG descriptors are based on orientation of gradients so rotation-invariance is out of discussion. The same applies for scale-invariance since the size of the scanning window is constant. When scale-invariance of HOG descriptors are our intention different techniques like different level(resolution) of pyramids of image can be used.

REFERENCES

[1] Navneet Dalal & Bill Triggs, "Histograms of Oriented Gradients for Human Detection", IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), Volume:1, 2005
[2] PAUL VIOLA & MICHAEL J. JONES, "Robust Real-Time Face Detection", International Journal of Computer Vision 57(2), 137–154, 2004
[3] http://www.mathworks.com/help/vision/ref/integralimage.html
[4] UCF - Center for Research in Computer Vision - Histograms of Oriented Gradients, http://crcv.ucf.edu/courses/CAP5415/Fall2012/Lecture-6a-Hog.pdf

SOURCE-CODE

test script.m

```
% Implementation of a variant of Histograms of Oriented Gradients (HOG) object
detector
% by Dalal&Trigggs.[1] Computing HOG algorithm efficiently using integral
gradient images,
% which are analogous to the standard integral image (Viola&Jones, Section 2).
[2]
%
% 1 - Read images (template & match)
% 2 - Calculate gradient histogram integrals & HOG distances
% 3 - Outline images with frames
%
% File: test_script.m
% Author: Evren KANALICI
% Date: 15/05/2016
% Computer Eng. - Computer Vision, Spring '16
% Yildiz Technical University
%
%

%% 1 - Read images (template & match)
RGBt = imread('car1.png');
It = im2double(rgb2gray(RGBt)); % template image
RGBm = imread('car2.png');
Im = im2double(rgb2gray(RGBm)); % match image

subplot(2,2,1);
imshow(RGBt);
subplot(2,2,2);
imshow(RGBm);
subplot(2,2,3);
imshow(It);
subplot(2,2,4);
imshow(Im);


%% 2 - Calculate gradient histogram integrals & HOG distances
QNUM=9; % number of phase quantization bins
NX=4; % number of x-subpatches in a window frame
NY=4; % number of y-subpatches in a window frame
TPATCH=[98 50 98+24 50+24]; % template patch frame (handled manually)
STEPSIZE=4; % step size for scanning

bbox=TPATCH;
ghistintegral=gradhistintegral(It,QNUM); % template gradient hist. integral
hog_template=hogintegral(ghistintegral,bbox,NX,NY); % template HOG desc.

ghistintegral=gradhistintegral(Im,QNUM); % match gradient hist. integral

[nrow, ncol] = size(Im);
rlen = bbox(3)-bbox(1); % frame row lenght
clen = bbox(4)-bbox(2); % frame col. lenght


rs=1:STEPSIZE:nrow-rlen;
cs=1:STEPSIZE:ncol-clen;
```

```
ds=zeros(size(rs,2),size(cs,2)); % HOG distances
for r=rs
    for c=cs
        bbox=[r c r+rlen c+clen]; % scanning frame

        % match image HOG desc.
        hog=hogintegral(ghistintegral,bbox,NX,NY);

        % HOG distances for current frame
        ds(ceil(r/STEPSIZE),ceil(c/STEPSIZE))=hogdistance(hog,hog_template);
    end
end

% print min. HOG distance
fprintf('min. distance [STEPSIZE:%d]: %.2f\n',STEPSIZE,min(min(ds)));


%% 3 - Outline images with frames
HOGTHRS=11; % histogram distance threshold

% outline template patch
overlay = drawframe(It,TPATCH(1),TPATCH(2),TPATCH(3),TPATCH(4));
subplot(1,2,1);
imshow(overlay);

[ix,iy]=find(ds<HOGTHRS); % indices for below threshold
n=size(ix,1);

% outline matched patches that below threshold
overlay=Im;
for i=1:n
    x1=(ix(i)-1)*STEPSIZE+1;
    y1=(iy(i)-1)*STEPSIZE+1;
    overlay=drawframe(overlay,x1,y1,x1+rlen,y1+clen);
end

fprintf('#%d patches found [HOG dist. threshold=%.2f]\n',n,HOGTHRS);
subplot(1,2,2);
imshow(overlay);
```

hogdistance.m

```
function [distance]=hogdistance(descA, descB)
%
% hogdistance - computes Euclidian distance between two HOG descriptions
% Input
%   descA: HOG description vector
%   descB: HOG description vector
% Output
%   distance: distance between two HOG description vectors.
%
% File: hogdistance.m
% Author: Evren KANALICI
% Date: 15/05/2016
% Computer Eng. - Computer Vision, Spring '16
% Yildiz Technical University
%
```

```
[nhist, qnum]=size(descA);
[nhistB, qnumB]=size(descB);
assert(nhist==nhistB && qnum==qnumB); % assert sizes are eqaul

% calculate euclidian distance between histograms for each patch
d=0;
for i=1:nhist
    d=d+sqrt(sum((descA(i,:)-descB(i,:)).^2));
end

distance=d;
end
```

hogintegral.m

```
function [hog]=hogintegral(ghistintegral,bbox,nx,ny)
%
% hogintegral - computes HOG descriptors for given image window
%
% Inputs:
%   ghistintegral: integral gradient histogram image
%   bbox: image window represented as [x1 y1 x2 y2]
%   nx,ny: number of (equal-size) descriptor cells along x and y window
dimensions
%
% Outputs:
%   hog: concat'ed 1D vector of HOG descriptions.
%
% File: hogintegral.m
% Author: Evren KANALICI
% Date: 15/05/2016
% Computer Eng. - Computer Vision, Spring '16
% Yildiz Technical University
%

% frame positions
x1=bbox(1);
y1=bbox(2);
x2=bbox(3);
y2=bbox(4);

% frame width & height
xstep = (x2-x1)/nx;
ystep = (y2-y1)/ny;

qnum=size(ghistintegral, 3); % bin size
desc=zeros(nx,ny, qnum); % HOG description matrix

for r=0:nx-1
    for c=0:ny-1
        % positions for integral image calculation
        p11 = [x1+(r+0)*xstep   y1+(c+0)*ystep];
        p12 = [x1+(r+0)*xstep   y1+(c+1)*ystep];
        p21 = [x1+(r+1)*xstep   y1+(c+0)*ystep];
        p22 = [x1+(r+1)*xstep   y1+(c+1)*ystep];
```

```
        % calculate histogram for the patch
        hist=ghistintegral(p11(1), p11(2), :) + ghistintegral(p22(1), p22(2), :)
...
          - ghistintegral(p12(1), p12(2), :) - ghistintegral(p21(1), p21(2), :);

        % l2-normalized histogram
        desc(r+1,c+1,:)=hist/norm(hist(:,:));
    end
end

% convert description to 1D of histograms
hog=reshape(desc,size(desc,1)*size(desc,2),size(desc,3));
```

gradhistintegral.m

```
function [retval]=gradhistintegral(img,qnum)
%
% gradimageintegral - computes gradient histogram integral image
%
% Inputs:
%   img: original gray-value image to process
%   qnum: number of gradient orientations bins
%
% Outputs:
%   retval: weighted integral gradient image of histograms
%
% File: gradhistintegral.m
% Author: Evren KANALICI
% Date: 15/05/2016
% Computer Eng. - Computer Vision, Spring '16
% Yildiz Technical University
%

[nrow, ncol] = size(img);

% calculate gradients
DX = diff(img,1,1);
DX = [zeros(1,ncol);DX]; % insert zero-row
DY = diff(img,1,2);
DY = [zeros(nrow,1) DY]; % insert zero-col

% gradient magnitudes
Igradmag = sqrt(DX .^2 + DY .^2);

% gradient phases
DX(DX==0) = eps('double'); % before atan calc.
Igradphase = atan(DY ./ DX);
[nnan, ~] = size(find(isnan(Igradphase)));
assert(nnan==0); % assert no NaN available

% quantize gradient orientations
Ij = zeros(nrow,ncol,qnum);
for r=1:nrow
    for c=1:ncol
        Ij(r,c,:) = quantize(qnum, Igradphase(r,c));
    end
end
```

```
% weight gradient orientations with magnitudes
Ij_weighted = zeros(nrow,ncol,qnum);
for q=1:qnum
    Ij_weighted(:,:,q) = Ij(:,:,q).*Igradmag;
end

retval = cumsum(cumsum(Ij_weighted,1),2); % integral of gradient histograms

end


function [q]=quantize(qnum,phase)
%
% quantize - quantize gradient orientations to histogram
%
% Inputs:
%   qnum: number of bins for histogram
%   phase: gradient orientation in rads. (could be negative)
%
% Ouputs:
%   q: quantized histogram of orientations (1D array of qnum size)
%

% compute orientation layers I1...n such that Ij has values 1 for all image
pixels
% with gradient orientation j and zeros otherwise
h = zeros(1, qnum);

% normalize phase (negative and zero values)
if phase<0
    phase=phase+pi;
elseif phase==0
    phase=eps;
end

step=pi/qnum; % step size of histogram
h(ceil(phase/step))=1; % place phase in histogram

q = h;

end
```

drawframe.m

```
function [overlay]=drawframe(I,x1,y1,x2,y2)
%
% drawframe - function to overlay image with frames defined with two corner
points
%
% Inputs:
%   I: Image to be processed
%   x1: top-left point x pos. of the frame
%   y1: top-left point y pos. of the frame
%   x2: bottom-right point x pos. of the frame
%   y2: bottom-right point y pos. of the frame
%
% Outputs:
%   overlay: overlay image with frames drawed
%
```

```
% File: drawframe.m
% Author: Evren KANALICI
% Date: 15/05/2016
% Computer Eng. - Computer Vision, Spring '16
% Yildiz Technical University
%
rows=size(I,1); % row count
cols=size(I,2); % column count

% x-positions & y-positions to iterate
xs=[x1:x2, ones(1,y2-y1)*(x2), x1:x2, ones(1,y2-y1)*x1];
ys=[ones(1,x2-x1)*y1, y1:y2, ones(1,x2-x1)*(y2), y1:y2];
assert(size(xs,2)==size(ys,2));


% IF any frame that does not fit inside the image THEN no-op
if any(xs>=rows) || any(ys>=cols) || any(xs<=1) || any(ys<=1)
    overlay=I;
    fprintf('[%d,%d %d%,d] frame that does not fit inside the image
\n',x1,y1,x2,y2);
    return
end

% iterate and mark
for i=1:size(xs,2)
    I(xs(i),ys(i))=1; % white pixels for overlay
end

overlay=I;
```