

Chris McCormick [About](#) [Tutorials](#) [Archive](#)

HOG Person Detector Tutorial

09 May 2013

One of the most popular and successful “person detectors” out there right now is the HOG with SVM approach. When I attended the Embedded Vision Summit in April 2013, it was the most common algorithm I heard associated with person detection.

HOG stands for Histograms of Oriented Gradients. HOG is a type of “feature descriptor”. The intent of a feature descriptor is to generalize the object in such a way that the same object (in this case a person) produces as close as possible to the same feature descriptor when viewed under different conditions. This makes the classification task easier.

The creators of this approach trained a Support Vector Machine (a type of machine learning algorithm for classification), or “SVM”, to recognize HOG descriptors of people.

The HOG person detector is fairly simple to understand (compared to SIFT object recognition, for example). One of the main reasons for this is that it uses

a “global” feature to describe a person rather than a collection of “local” features. Put simply, this means that the entire person is represented by a single feature vector, as opposed to many feature vectors representing smaller parts of the person.

The HOG person detector uses a sliding detection window which is moved around the image. At each position of the detector window, a HOG descriptor is computed for the detection window. This descriptor is then shown to the trained SVM, which classifies it as either “person” or “not a person”.

To recognize persons at different scales, the image is subsampled to multiple sizes. Each of these subsampled images is searched.

Original Work

The HOG person detector was introduced by Dalal and Triggs at the CVPR conference in 2005. The original paper is available [here](#).

The original training data set is available [here](#).

Gradient Histograms

The HOG person detector uses a detection window that is 64 pixels wide by 128 pixels tall.

Below are some of the original images used to train the detector, cropped in to the 64x128 window.

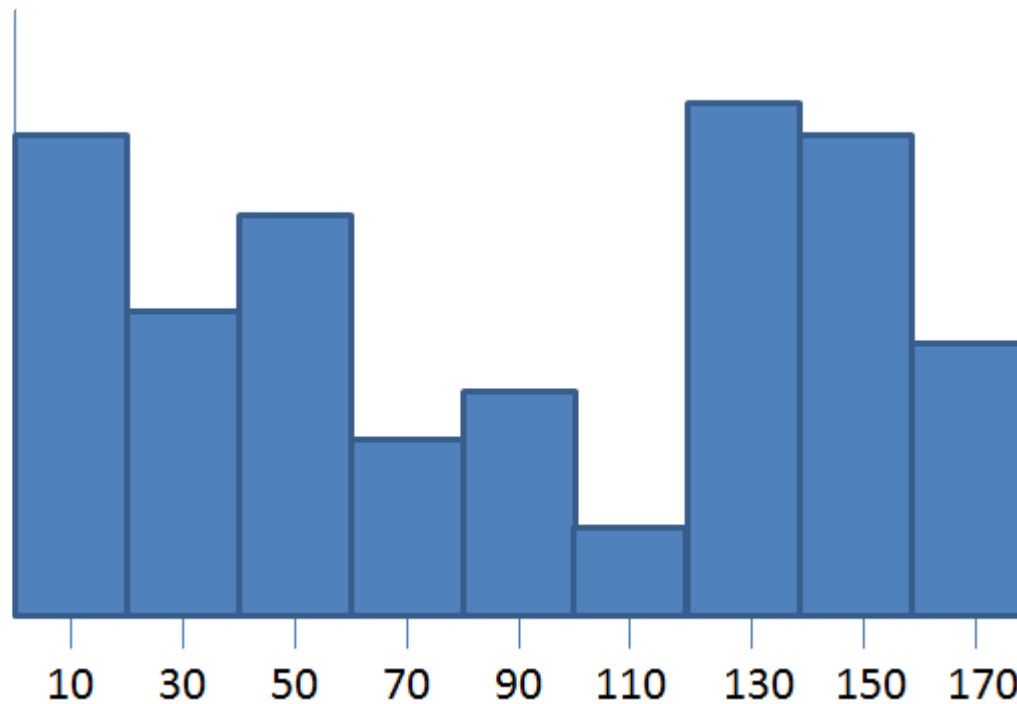


To compute the HOG descriptor, we operate on 8x8 pixel cells within the detection window. These cells will be organized into overlapping blocks, but we'll come back to that.

Here's a zoomed-in version of one of the images, with an 8x8 cell drawn in red, to give you an idea of the cell size and image resolution we're working with.



Within a cell, we compute the gradient vector at each pixel (see my post on [gradient vectors](#) if you're unfamiliar with this concept). We take the 64 gradient vectors (in our 8x8 pixel cell) and put them into a 9-bin histogram. The Histogram ranges from 0 to 180 degrees, so there are 20 degrees per bin.



Note: Dalal and Triggs used “unsigned gradients” such that the orientations only ranged from 0 to 180 degrees instead of 0 to 360.

For each gradient vector, it’s contribution to the histogram is given by the magnitude of the vector (so stronger gradients have a bigger impact on the histogram). We split the contribution between the two closest bins. So, for example, if a gradient vector has an angle of 85 degrees, then we add 1/4th of its magnitude to the bin centered at 70 degrees, and 3/4ths of its magnitude to the bin centered at 90.

I believe the intent of splitting the contribution is to minimize the problem of gradients which are right on the boundary between two bins. Otherwise, if a

strong gradient was right on the edge of a bin, a slight change in the gradient angle (which nudges the gradient into the next bin) could have a strong impact on the histogram.

Why put the gradients into this histogram, rather than simply using the gradient values directly? The gradient histogram is a form of “quantization”, where in this case we are reducing 64 vectors with 2 components each down to a string of just 9 values (the magnitudes of each bin). Compressing the feature descriptor may be important for the performance of the classifier, but I believe the main intent here is actually to generalize the contents of the 8x8 cell.

Imagine if you deformed the contents of the 8x8 cell slightly. You might still have roughly the same vectors, but they might be in slightly different positions within the cell and with slightly different angles. The histogram bins allow for some play in the angles of the gradients, and certainly in their positions (the histogram doesn't encode *_where_* each gradient is within the cell, it only encodes the “distribution” of gradients within the cell).

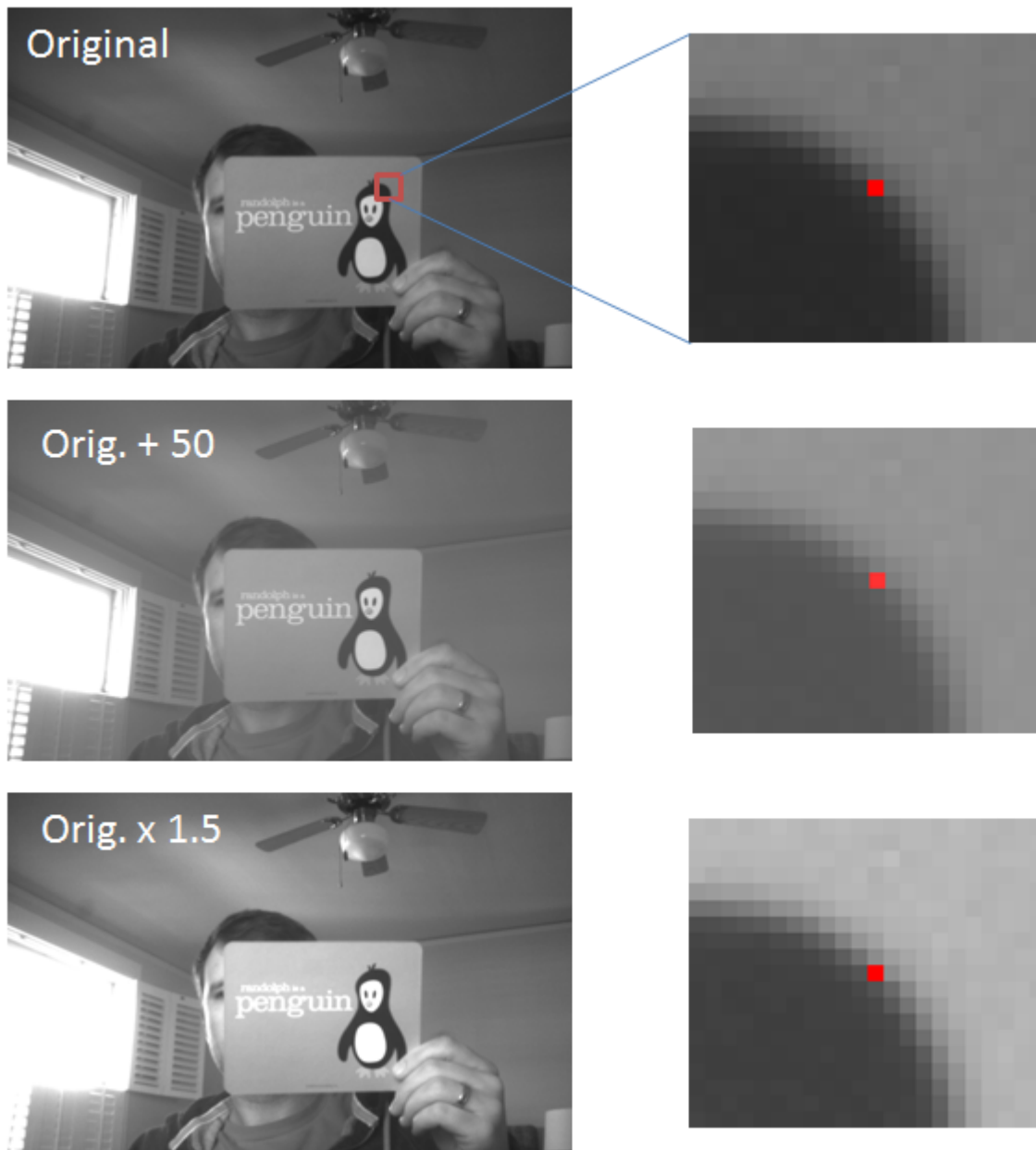
Normalizing Gradient Vectors

The next step in computing the descriptors is to normalize the histograms. Let's take a moment to first look at the effect of normalizing gradient vectors in general.

In my post on [gradient vectors](#), I show how you can *add or subtract* a fixed amount of brightness to every pixel in the image, and you'll still get the same

the same gradient vectors at every pixel.

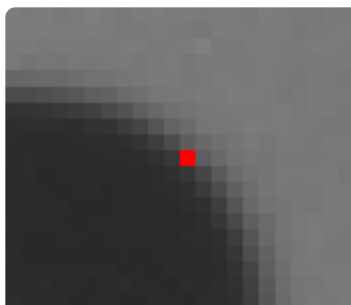
It turns out that by normalizing your gradient vectors, you can also make them invariant to *multiplications* of the pixel values. Take a look at the below examples. The first image shows a pixel, highlighted in red, in the original image. In the second image, all pixel values have been increased by 50. In the third image, all pixel values in the original image have been multiplied by 1.5.



Notice how the third image displays an increase in contrast. The effect of the multiplication is that bright pixels became much brighter while dark pixels only

became a little brighter, thereby increasing the contrast between the light and dark parts of the image.

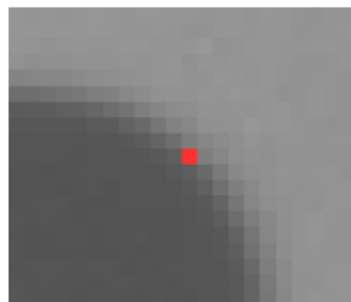
Let's look at the actual pixel values and how the gradient vector changes in these three images. The numbers in the boxes below represent the values of the pixels surrounding the pixel marked in red.



		93	
56			94
		55	

$$\nabla f = \begin{bmatrix} 38 \\ 38 \end{bmatrix}$$

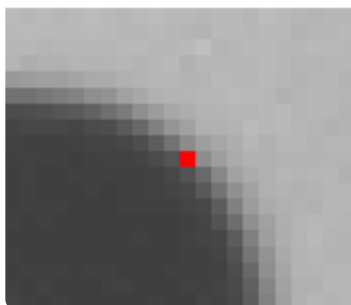
$$|\nabla f| = \sqrt{(38)^2 + (38)^2} = 53.74$$



		143	
106			144
		105	

$$\nabla f = \begin{bmatrix} 38 \\ 38 \end{bmatrix}$$

$$|\nabla f| = \sqrt{(38)^2 + (38)^2} = 53.74$$



		140	
84			141
		83	

$$\nabla f = \begin{bmatrix} 57 \\ 57 \end{bmatrix}$$

$$|\nabla f| = \sqrt{(57)^2 + (57)^2} = 80.61$$

The gradient vectors are equivalent in the first and second images, but in the third, the gradient vector magnitude has increased by a factor of 1.5.

If you divide all three vectors by their respective magnitudes, you get the same result for all three vectors: $[0.71 \ 0.71]'$.

So in the above example we see that by dividing the gradient vectors by their magnitude we can make them invariant (or at least more robust) to changes in contrast.

Dividing a vector by its magnitude is referred to as normalizing the vector to unit length, because the resulting vector has a magnitude of 1. Normalizing a vector does not affect its orientation, only the magnitude.

Histogram Normalization

Recall that the value in each of the nine bins in the histogram is based on the magnitudes of the gradients in the 8x8 pixel cell over which it was computed. If every pixel in a cell is multiplied by 1.5, for example, then we saw above that the magnitude of all of the gradients in the cell will be increased by a factor of 1.5 as well. In turn, this means that the value for each bin of the histogram will also be increased by 1.5x. By normalizing the histogram, we can make it invariant to this type of illumination change.

Block Normalization

Rather than normalize each histogram individually, the cells are first grouped into blocks and normalized based on all histograms in the block.

The blocks used by Dalal and Triggs consisted of 2 cells by 2 cells. The blocks have “50% overlap”, which is best described through the illustration below.



This block normalization is performed by concatenating the histograms of the four cells within the block into a vector with 36 components (4 histograms x 9 bins per histogram). Divide this vector by its magnitude to normalize it.

The effect of the block overlap is that each cell will appear multiple times in the final descriptor, but normalized by a different set of neighboring cells. (Specifically, the corner cells appear once, the other edge cells appear twice each, and the interior cells appear four times each).

Honestly, my understanding of the rationale behind the block normalization is still a little shaky. In my earlier normalization example with the penguin flash card, I multiplied every pixel in the image by 1.5, effectively increasing the contrast by the same amount over the *whole* image. I imagine the rationale in the block normalization approach is that changes in contrast are more likely to

occur over smaller regions within the image. So rather than normalizing over the entire image, we normalize within a small region around the cell.

Final Descriptor Size

The 64 x 128 pixel detection window will be divided into 7 blocks across and 15 blocks vertically, for a total of 105 blocks. Each block contains 4 cells with a 9-bin histogram for each cell, for a total of 36 values per block. This brings the final vector size to 7 blocks across x 15 blocks vertically x 4 cells per block x 9-bins per histogram = 3,780 values.

HOG Detector in OpenCV

OpenCV includes a class for running the HOG person detector on an image.

Check out [this post](#) for some example code that should get you up and running quickly with the HOG person detector, using a webcam as the video source.


HOG Descriptor in Octave / MATLAB


To help in my understanding of the HOG descriptor, as well as to allow me to easily test out modifications to the descriptor, I've written a function in Octave for computing the HOG descriptor for a 64x128 image.




As a starting point, I began with the MATLAB code provided by another researcher [here](#). That code doesn't implement all of the features of the original


HOG person detector, though, and didn't make very effective use of vectorization.


I've dedicated a separate post to the Octave code, check it out [here](#).




49 Comments [mccormickml.com](#) 

 Recommend 19  Tweet  Share Sort by Best ▾



LOG IN WITH OR SIGN UP WITH DISQUS 



Casey Cessnun • 10 months ago

Chris, firstly, thank you very much for the series of articles you've written here. Fantastic resource. Secondly, apologies for being obtuse. However, I am confused over this section: 'For each gradient vector, it's contribution to the histogram is given by the magnitude of the vector (so stronger gradients have a bigger impact on the histogram). We split the contribution between the two closest bins. So, for example, if a gradient vector has an angle of 85 degrees, then we add 1/4th of its magnitude to the bin centered at 70 degrees, and 3/4ths of its magnitude to the bin centered at 90.'

How do I know, based on the angle, how much of the magnitude to assign where?

Thank you Very Much!

-Casey

1 ^ | ▾ • Reply • Share ›



Wing Poon • 2 years ago

Thanks for the writeup. Per Dalal in his ACM talk on YouTube, the block-level normalization is to deal with situations whereby part of the body is in the shade while the other is in the sun. On a different matter, interesting that in that talk, he showed a chart whereby he empirically determined that a 6x6 cell size coupled with a 3x3 block size produced the lowest errors (i.e. not 8x8/2x2).

1 ^ | v • Reply • Share ›



Chris McCormick Mod ➔ Wing Poon • 2 years ago

Interesting! Yeah, when you have something in the sun and something in the shade in a photograph, you call that high contrast, so block normalization is intended to deal with contrast.

That's interesting, too, about the parameters--you should be able to try those settings out in most implementations of HOG, including my Matlab one.

^ | v • Reply • Share ›



Yudhi Anggara • 9 months ago

hello mr Chris, im currently doing my last college project with this HOG method, i want to ask about the orientation binning, why HoG uses 9 bins? can we actually change the amount of bins to 10/15/20?, is there a reason for using 9 bins?

:: my professor asked me this question and i was like 'dead-end'.

^ | v • Reply • Share ›



Bamini Thavarajah • a year ago

How we can get HOG feature form one cell only using above code?

^ | v • Reply • Share ›



Arif Shakil • a year ago

Chris hi, I'm using your tutorial to help in my thesis. I got the overall idea on how it is working (thanks to your awesome tutorial), but just stuck at one place. In the Image/Validation folder, a CSV file is associated with each image. What is it and how does that work? I am unable to use it on my images, however the samples you gave works fine.

^ | v • Reply • Share ›

[^](#) | [v](#) · [Reply](#) · [Share](#) >**Arif Shakil** → Arif Shakil · a year ago

I'm having trouble to just run the code... i just realized it. I have no idea how it works. I previously thought I did. :|

[^](#) | [v](#) · [Reply](#) · [Share](#) >**Lam Lam** · 2 years ago

I have some suggestion and questions about your tutorial first of all how can we visuliazze the normalized 9-bin histogram which is 8x8 blocks(3780 elements) when we moving the 16X16 blocks windows 8 pixels away should we take the operated value form previous normalized 9-bin histogram in second operation?(I take the raw value)

For more details:

In first part of finding gradient x and y the kernel or convolution matrix called Prewitt

In block normalization actually provide four different methods

Gradient and direction actually form a vector(the arrows)

If this tutorial can go through more about block normalization, SVM and the weight of voting in histogram (magnitude affect weighting) it will be better

[^](#) | [v](#) · [Reply](#) · [Share](#) >**김범석** · 2 years ago

Hello, Chris.

I wanna find the person using your opensource.

But My research is first step. So I need to simple method.

Using your source, I want to find a person by single scale.

Your method is using the multiscale. I checked in serchImage.m.

Please how to revise code find the person by single scale.

Thank you. I always thank you.

[^](#) | [v](#) · [Reply](#) · [Share](#) >



Chris McCormick Mod → 김범석 • 2 years ago

Try modifying the 'scaleRange' variable [here](#).

^ | v • Reply • Share ›



Nikhil Pandey • 2 years ago

I trained and used a HOG detector but tbh, till today, I was just speculating what the hell I was doing. It is well written and nicely structured. Thanks.

^ | v • Reply • Share ›



Chris McCormick Mod → Nikhil Pandey • 2 years ago

Haha, yeah, that's often how it goes. Thanks for the kind words!

^ | v • Reply • Share ›



azkya c • 2 years ago

Hai Chris. i downloaded and tried your own matlab [code.it](#) was great!! then i was wondering. why did you cropped the images into 8kb ? :)

^ | v • Reply • Share ›



Chris McCormick Mod → azkya c • 2 years ago

Since the detector window is 64 x 128, the smallest people in the image don't need to be any larger than 128 pixels tall. So you can compress the image size down, and this also reduces the number of scales that you have to search at.

^ | v • Reply • Share ›



azkya c → Chris McCormick • 2 years ago

okaay. i got it. thanks :)

^ | v • Reply • Share ›



Bhakti • 2 years ago

If is it Possible then send MATLAB code of HOG for Car detection please.....thank you

^ | v • Reply • Share ›



No_one → Bhakti • 2 years ago



Look here Bhakti: <http://www.vlfeat.org/>

^ | v · Reply · Share ›



Chris McCormick Mod → Bhakti · 2 years ago

I don't have that, sorry!

^ | v · Reply · Share ›



Wing Poon · 2 years ago

What overlap (e.g. 50%?) is recommended when sliding the detection window to achieve a good compromise between computation effort vs accuracy? I recall Dalal recommending a scaling ratio of 1.25x between passes.

^ | v · Reply · Share ›



Chris McCormick Mod → Wing Poon · 2 years ago

I'm not experienced enough with using the detector in practice to offer any insight there, sorry!

^ | v · Reply · Share ›



Niki · 2 years ago

Hi Chris!

Thank for such a nice tutorial! I have a quick question. Using your code, I have wrote the getHistogram function in python. When all the laftBinIndex that are zero are changed to the number of bins (in this case 9) to compensate for the wrap up effect, at the end there will be no contribution in the first bin (with leftBinIndex=1 and rightBinIndex=2). In fact, when I test the code on an image, I always get zero for the first bin. I have checked my code and I don't see anything wrong, as I mentioned I basically implemented your code in python. I was wondering if you can give me a hint on something that might not be implemented correctly.

Thank you so much!

^ | v · Reply · Share ›



Chris McCormick Mod → Niki · 2 years ago

Thanks, Niki!

Hmm, I remember that was a tricky piece of the code! Simplest thing to check, first, though...Matlab arrays are indexed from 1 while Python arrays are indexed from 0!

though Matlab arrays are indexed from 1 while Python arrays are indexed from 0:

Maybe the problem is related to that?

^ | v · Reply · Share ›



Niki → Chris McCormick · 2 years ago

Thank you for your response! That's right, instead of putting zero, I put it equal to 1 in the python code.

^ | v · Reply · Share ›

[Show more replies](#)



CrazyBrazilian · 2 years ago

Hi Chris !

Very nice post ! I found your explanation so easy to understand. I have a quick question and I was hoping you could help me. I watched the video by prof. Mubarak on youtube. He shows a plotting of the image using the histogram.

Once you have calculated the whole histogram, do you know how to use that to plot the image that resembles the human in the original image.

Thank you so much !

^ | v · Reply · Share ›



Chris McCormick Mod → CrazyBrazilian · 2 years ago

Thanks! Glad it was helpful.

I haven't played much with visualization of HOG descriptors.

I know that the VLFeat HOG implementation has some support for this, as seen here:

<http://www.vlfeat.org/overv...>

Good luck!

^ | v · Reply · Share ›



Glassies Adamson · 2 years ago

Hi! I was wondering if it could still detect images of person if it is half body or top view only? Can you help me with this, please. Thank you so much :D

^ | v · Reply · Share ›



Chris McCormick Mod → Glassies Adamson • 2 years ago

I think this is a weakness of the original HOG detector--the person can't be too occluded for it to work well. There was some later work done that used gradient histograms but with a connected components model--I think this was supposed to handle occlusion better.

^ | v • Reply • Share ›



aamer aamer • 2 years ago

can u provide some code in python opencv like the one provided in c++

^ | v • Reply • Share ›



Bhakti • 3 years ago

Here You mention the detection window is 64 pixel wide and 128 pixel tall. so my question is the can I changes in Detection window size ? Here u can extract feature of Person. but instead of person I have use Car then its Detection window is in Horizontal form...like 128 pixel wide and 64 pixel tall...so What is the answer of my question....Thank you So much...

^ | v • Reply • Share ›



Chris McCormick Mod → Bhakti • 2 years ago

Sorry for the delayed response, but yes, absolutely. The HOG detector has been modified to detect cars and bicycles--you can definitely play with the dimensions of the detection window.

^ | v • Reply • Share ›



Bhakti → Chris McCormick • 2 years ago

thank you so much

^ | v • Reply • Share ›



Tobias Becker • 3 years ago

Your description helped a lot—in contrast to the original paper, which is quite terse when it comes to the implementation. I have one question though: What do I do when my x-gradient is zero? Because division naturally would blow up. And when the y-gradient is not zero there's definitely some magnite to vote with.

^ | v · Reply · Share ›



Tobias Becker → Tobias Becker · 3 years ago

Oh, I just solved it. Using atan2 instead of dividing yourself makes life easier.

^ | v · Reply · Share ›



Chris McCormick Mod → Tobias Becker · 3 years ago

Cool; I used atan2 in my code as well.

Glad the article was helpful, thanks!

^ | v · Reply · Share ›



Tapas · 3 years ago

Thanks for nice explanation.

^ | v · Reply · Share ›



Chris McCormick Mod → Tapas · 3 years ago

Thanks, glad it was helpful!

^ | v · Reply · Share ›



Vadim Timoftica · 3 years ago

Hi people how convert HOGDescriptor hog; to int?

^ | v · Reply · Share ›



Chris McCormick Mod → Vadim Timoftica · 3 years ago

Do you mean converting the floating point values in the descriptor to integers?

^ | v · Reply · Share ›



Alvaro Gregorio Gómez · 3 years ago

Really clarifies a lot the paper, I've been searchin for this for one month in order to do my bachelor thesis. I was wondering how much time it requires to train a decent classifier

^ | v · Reply · Share ›



Yutian Li · 3 years ago



Thanks for the illustration and explanation. That makes much more sense to me. I feel sometimes the paper tries to be generic and gloss over implementation details.

^ | v · Reply · Share ›



Chris McCormick Mod ➔ Yutian Li · 3 years ago

Thanks, glad it was helpful!

^ | v · Reply · Share ›



Raymond Phan · 3 years ago

Very good explanation on HOG. The original paper was somewhat confusing and the material I've found on Wikipedia and other Computer Vision course websites was rather unclear. This is the first treatise on the topic that is simple to understand and as a bonus you've included MATLAB / Octave code. Thank you for your efforts!

^ | v · Reply · Share ›



Chris McCormick Mod ➔ Raymond Phan · 3 years ago

Thanks for the kind words, so glad to hear it was helpful!

1 ^ | v · Reply · Share ›



Raymond Phan ➔ Chris McCormick · 3 years ago

No problem at all. If I may suggest something, inside your `getHistogram.m` code where you're calculating the contribution of the magnitudes per bin, I see you've computed the dot product between the weight vector and the magnitude for both the left and right bin contributions. After, you use the `sum` function. Because you are transposing the weight vector so that it becomes a row (i.e. `leftPortion(pixels)'`) and with the multiplication with the magnitudes being a column, this already calculates the sum of products (i.e. the dot product) for you and so `sum` is not necessary to be called... actually it's redundant. This obviously provides little computational boost but it's something to consider. Or if you wish the sum to remain, explicitly do an element-wise product of the two vectors while maintaining that the weight vector is a column (i.e. `sum(leftPortion(pixels) .* magnitudes(pixels));`). Otherwise, great code!

^ | v · Reply · Share ›



M · 3 years ago



ms • 3 years ago

Love how you make it look so simple! Thanks allot!

^ | v • Reply • Share ›



Sidharth Bhagwan Bhorge • 3 years ago

very nice explanation sir

^ | v • Reply • Share ›



Sanjaya Nayak • 3 years ago

Great explanation Sir, Thank you.

^ | v • Reply • Share ›



Ahmed • 3 years ago

Thank you.

Simple and neat

^ | v • Reply • Share ›



Faraz • 3 years ago

Fantastic post, thank you.

^ | v • Reply • Share ›

ALSO ON MCCORMICKML.COM

Stereo Vision Tutorial - Part I

29 comments • 3 years ago



Chris McCormick — Hi, Akhil - I think some errors are to be expected, especially if there's not a lot of visual distinction in the image. ...

DBSCAN Clustering

1 comment • 2 years ago



Anuj Choudhury — I couldnt understand a detail.The growCluster function has an if else if statement.Why is the else statement in it ...

AdaBoost Tutorial

17 comments • 3 years ago



disqus_OCqj7YZHey — Good morning,Thanks for your early reply Chris! If you now some sources of information about this type of ...

Image Derivative

16 comments • 3 years ago



Aya al-bitar — I've read your blogs about HOG , Gradient vector and image derivative and they were extremely helpful, Thanks ..

Related posts

[The Inner Workings of word2vec](#) 12 Mar 2019

[Applying word2vec to Recommenders and Advertising](#) 15 Jun 2018

[Product Quantizers for k-NN Tutorial Part 2](#) 22 Oct 2017

© 2019. All rights reserved.