Unknown Title

Skillcate AI : : 8/13/2022

SkillC

Skillcate AI

Aug 13, 2022

.

8 min read

.

Histogram of Oriented Gradients (HOG) — Simplest Intuition

> Not all data is created equal. Some of it is structured and the vast majority is unstructured.

Just to give you a perspective, by this year 2022, around 93% of the world's data would be unstructured. Audio files, image /video files, emails and twitter/facebook posts are all unstructured data.

Challenge with unstructured data is that it requires significant *feature extraction* efforts to convert it to a representation that could be used for building intelligent models.

Watch the video tutorial instead

We have done a video tutorial on this project. If you are more of a video person, go ahead and watch it on youtube. FYI, we launch new machine learning projects every week. So, make sure to subscribe to our channel to get access to all of our free ML courses.

Introduction to feature extraction — from Image data

To start with, let's take a simple example. In this B&W image, clearly we may observe that it's a digit 8. But the question is, how would a computer know that it's a no 8? The answer lies in how a computer stores this image data, which is basically a matrix. This particular image has a size of 22 x 16, meaning there are 22 pixels along the height and 16 pixels along the width. And, in totality, there are 352 pixels.

B&W Image, what we see

How computer stores it

0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

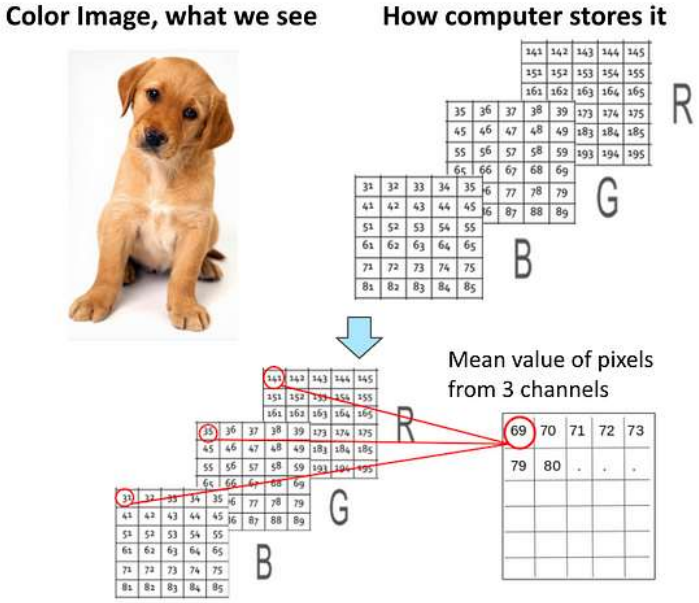
- No. of pixel (22 x 16): 352
- Pixel value for black: ~0
- Pixel value for white: ~255
- Features of this image data:

Feature	Label
F1, F2, F3,..., F352	8

A computer stores a number between 0 & 255 for each of these 352 pixels, which is basically a measure of the respective pixel's brightness. As you may observe yourself, brighter pixels have a value approaching 255 and darker pixels have values approaching 0.

> With this understanding, a very simple way to represent features for this image is to append all these rows horizontally and have a 1 x 352 matrix.

Alright, that's about B&W images. Things would change slightly for a color image. For color RGB images, there are three pixel matrices, one each for red, green and blue color intensities (again on a scale of 0–255).



- No. of pixel (22x16x3): 1056
- 3 pixel matrices; one each for Red/Green/Blue
- Pixel value for RGB: 0-255
- Features of this image data:

Feature	Label
F1, F2, F3,..., F352	Dog

We may replicate the previously discussed approach for B&W image, here as well. For that we simply need to prepare a new pixel matrix carrying mean pixel values for corresponding pixels from the three matrices, which are called as channels. Further steps are the exact same as discussed previously.

While this approach of feature extraction from images is simple, it is prone to a lot of problems. Consider we are comparing these two images. We have extracted features for these using our simple approach.

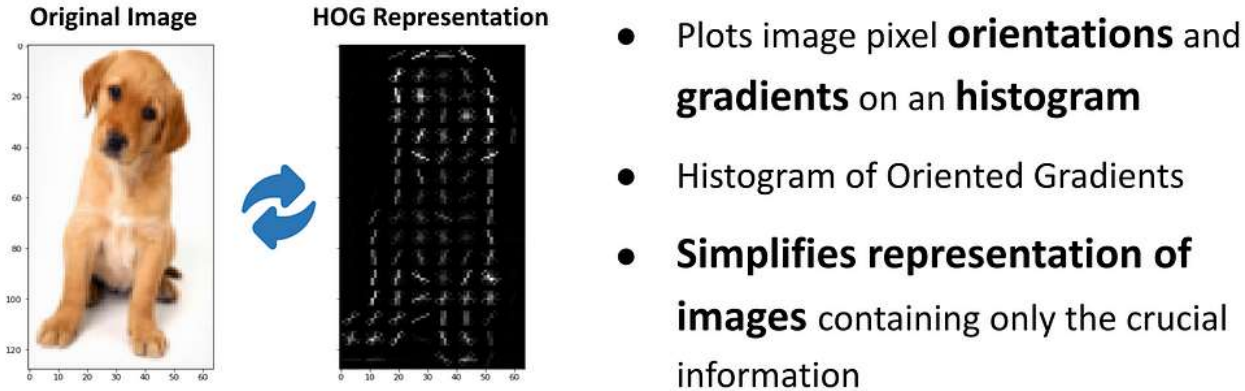
- As we may observe, this second image has a background, while the first does not. Also, the color of the two dogs are very different. And not just that, even the expressions of the two dogs are different. All these factors are collectively called as A good feature extraction approach would tend to minimise this noise, and clearly our simple approach can not do it, as it directly uses image pixel intensities
- Not just that, our simple approach is , as it retains lot of unnecessary information, that a good feature extraction approach may otherwise drop

  Hence, we need a more systematic approach to extract features from images, an approach that retains only the crucial information, while discarding the rest.

For achieving this, we use HOG or Histogram of Oriented Gradients.

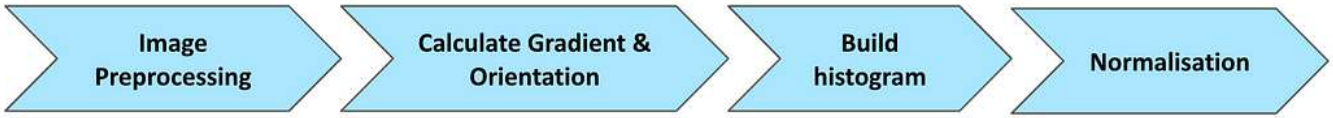
Introduction to HOG

Simply put, HOG computes pixel-wise gradients and orientations, and plots them on a histogram. Hence the name, histogram of oriented gradients. For this original image, this is the HOG representation.



As we may observe, HOG has simplified representation of this image drastically, retaining only a small piece of information. This way, HOG is able to minimise overall noise. Now, let's go step-by-step into building this HOG representation for this original image, where I would try to cover the indepth intuition without going too much into technicalities.

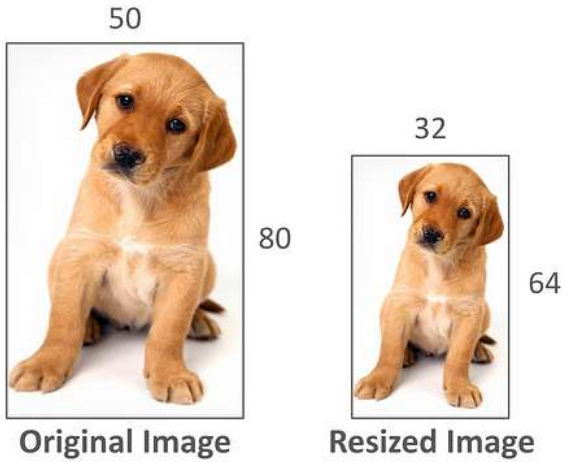
Step-by-step HOG walkthrough



When we are building a face recognition machine learning model, we are essentially feeding tons of images for training. And quite naturally, the first step is to ensure all images are of equal size.

Image preprocessing

As first step here, we perform image preprocessing, wherein we standardize image size (to 32x64 in this example). This means we have 32 pixels along image width and 64 pixels along image height.

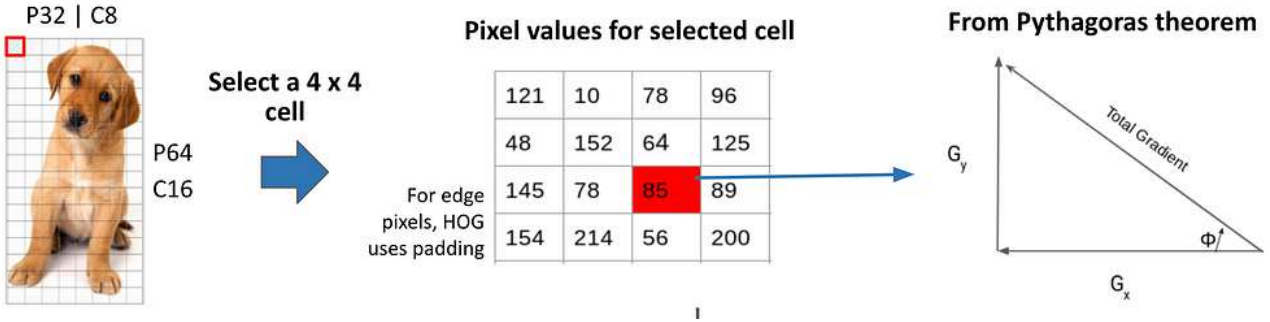


- Resized image size: 32x64
- HOG Cell size: 4x4
- Recommended:
  - Image size: 64x128
  - cell size: 8x8 / 16x16

Calculate Gradient & Orientation

As we already know, HOG is histogram of oriented gradients, in this section we would calculate the gradient and orientation, which we would then plot on a histogram in the next section.

Here, we have divided our image into 4x4 cells. As you may see, we have 8 such cells along the width, denoted as C8 and 16 cells along the height, denoted as C16.



For this first cell, let's assume this is how the pixel value matrix is like. As you may observe, this is 4x4, the way we discussed previously.

The first thing we have to calculate here is gradient in x and y directions, that we call, Gx and Gy. Gradient in x direction, or Gx is the difference of pixel value on the right of this highlighted pixel to the pixel value on the left. So, this is 89–78 = 11. Alright, this is our Gx. Similarly, Gradient in y direction, or Gy is the difference of pixel value above this highlighted pixel to the pixel value below. So, this is 64–56 = 8. This is our Gy.

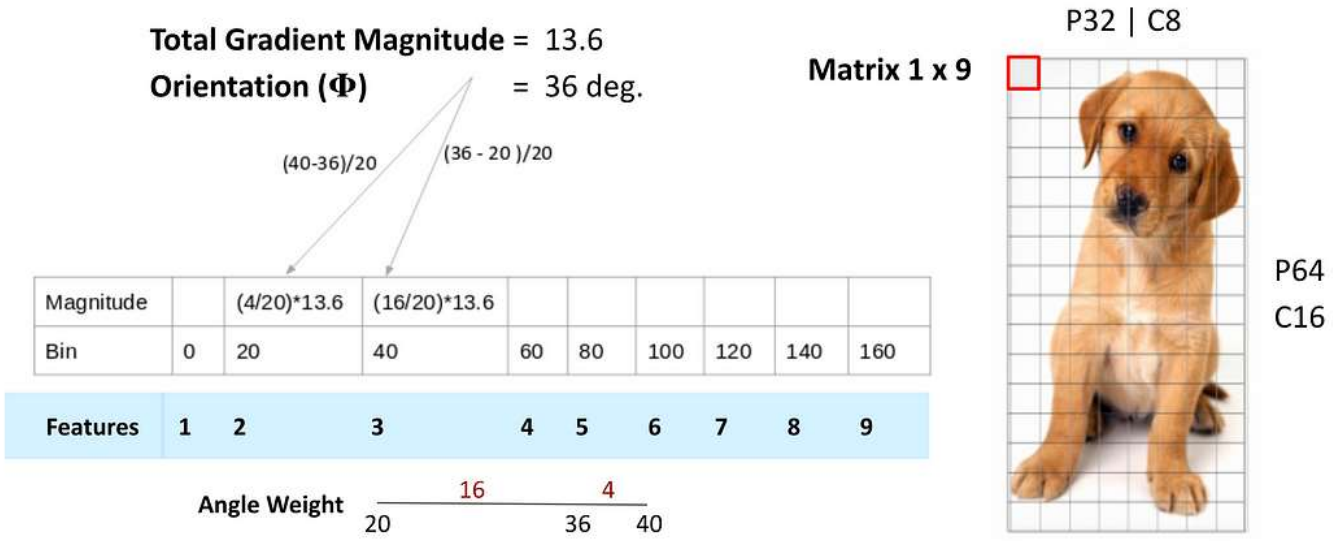
Now, to calculate the total gradient magnitude and orientation from these Gx and Gy gradient components, we would use pythagoras theorem (yes the one we studied in school). So, the total gradient magnitude is the root sum of squares of Gx and Gy, which is 13.6. Similarly, orientation is the inverse tangent of Gy by Gx, which is 36 degrees.

Similarly, HOG algorithm would compute gradients and orientations for all these pixels of this selected 4x4 cell. For the image edge pixels, HOG uses a technique called padding. **Please refer to the link in the description part of this video to know more on padding.**

As a side note: recommended practice is to resize images to 64x128 and keep cell size as 8x8 or 16x16. But in our case, the input images themselves are very small. So, we are using small resize and cell size values.

Build histogram

Next up, let's try to plot these gradient and orientation values on a histogram. Gradient and orientations values for our highlighted pixel are 13.6 and 36 degrees respectively. We already know that this orientation value may vary between 0 to 180 degrees. For the histogram, HOG prepares bins of 20 degrees each. So, there are 9 bins. Next up, HOG would start inserting gradient magnitude values as per the pixel orientation into these 9 bins.



For our highlighted pixel, orientation is 36 degrees which is closer to 40 degrees. So the major contribution of gradient magnitude would go to the 40 degree bin and the minority would go into the 20 degrees bin. We use angle weights to do it. Simply put, 16/20 is the weight that is assigned to the 40 degree bin, which is the major chunk of gradient magnitude and 4/20 is the weight assigned to the 20 degree bin, which is minor.

Similarly, all other gradient values are added to this histogram based on their pixel orientations for this selected cell. Towards, the end we get a 1x9 feature matrix for this cell.

Then HOG would compute such 1x9 feature matrices for the remaining cells. Once done, next step is normalisation.

Normalisation

Localised gradients of image are sensitive to overall lighting. We can partly reduce this variation by normalising the gradients. For this, we make groups of cells, known as blocks. Here, we have grouped 4 cells into each block. Thus, we have a total of 7 blocks along the width and 15 blocks along the height, when we shift this block by one cell at a time. Like this.

Alright! For a cell, HOG previously prepared feature matrices of 1 x 9 each. Typically, a normalized matrix for a block would have cell features appended horizontally. So for a block, a normalised matrix would be 1 x 36.

Now to get a normalized vector for this block, we need to compute this k, which is the root sum of squares of all 36 block features. Then we divide all features with this computed k to finally reach to our normalized vector for this highlighted first block.

- Gradients for highlighted Block (having 36 features):

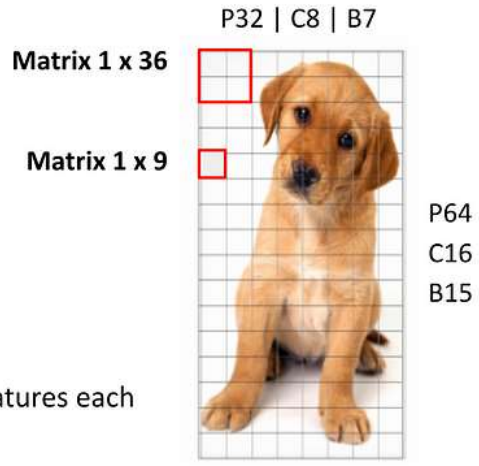
$$V = [a_1, a_2, a_3, \ldots a_{36}]$$

- Root of the sum of squares:

$$k = \sqrt{(a_1)^2 + (a_2)^2 + (a_3)^2 + \ldots + (a_{36})^2}$$

$$\text{Normalised Vector} = \left[\frac{a_1}{k}, \frac{a_2}{k}, \frac{a_3}{k}, \ldots, \frac{a_{36}}{k}\right]$$

- Normalised vectors created for all 105 blocks, having 36 features each
- Image **Feature Descriptor: 1 x 3780 matrix**



Similar way, HOG would compute normalized feature vector for all 105 blocks, by moving this block one cell at a time, both along this image width and height.

Once done, HOG appends the 36 features from all 105 block normalised vectors horizontally, giving us a 1 x 3780 dimensional image descriptor. We could validate this number during the coding part.

Well, this completes intuition on HOG feature descriptor.

By the way, even 3780 are a lot of features, which would make our model building part computationally expensive and might lead to overfitting. So we need to reduce these dimensions, for which we could use a popular dimensionality reduction technique called Principal Component Analysis, aka, PCA.

Brief about Skillcate

At Skillcate, we are on a mission to bring you application based machine learning education. We launch new machine learning projects every week. So, make sure to subscribe to our youtube channel and also hit that bell icon, so you get notified when our new ML Projects go live.

Shall be back soon with a new ML project. Until then, happy learning 🤗!!

Enjoy the read? Reward the writer.Beta

Your tip will go to Skillcate AI through a third-party platform of their choice, letting them know you appreciate their story.