Histograms of Oriented Gradients

MESKI Kaddour Seyf Eddine
University of the Balearic Islands
Palma, Spain
seyfeddinemsk@gmail.com

1 How does HOG work?

You're already able to detect people in a video stream, and you can stop here if you wish! But if you want to know how the algorithm works, read on, I've tried to explain it in simple terms. To understand how HOG (**Histograms of Oriented Gradients**) works, it's necessary to first understand what is a gradient, and what is an histogram.

1.1 What is a gradient?

For a 1D function ***f*** depending on variable ***x***, the gradient is simply the derivative of the function. At a given point, the derivative gives the local slope of the function. For example, let's consider the function ***f(x) = x***. The derivative of this function is ***f'(x)*** = 1. So this function is a straight line, and its derivative is a constant, which means that the slope is a constant.

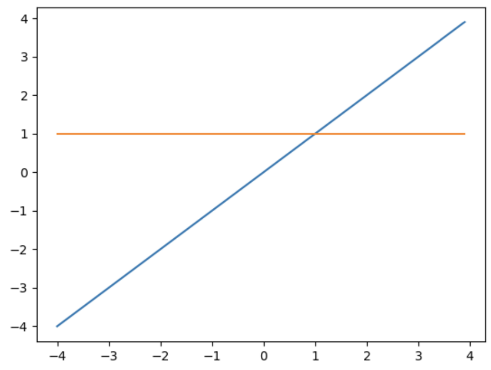

Figure 1: f(x) = x and it's derivative, f'(x) = 1

If we now consider ***f(x)*** = x^2, the derivative is ***f(x) = 2x***. For ***x***<0, the derivative is negative and the function is decreasing. For ***x***>0, the derivative is positive and the function is increasing. For ***x*** = 0, the derivative is 0. This means that the local slope is 0. In other words, here, the function is locally flat.

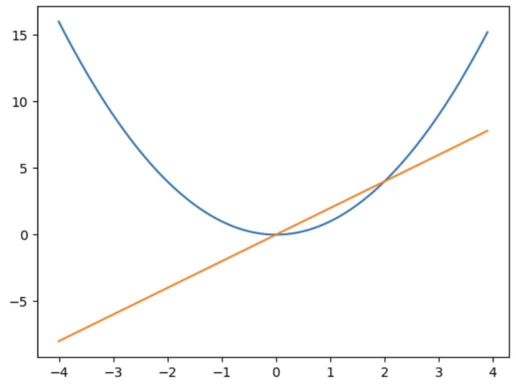

Figure 2: f(x) = x^2 and its derivative, f'(x) = 2x

In 2D, in the ***(x,y)*** plane, a function of ***x*** and y is a surface giving the altitude at every point. The gradient is the generalization of the derivative: at a given ***(x,y)*** point, the gradient is oriented towards the direction of maximum slope, and its magnitude is the slope of the plane tangent to the surface at this point. Here is an illustration from the ***wikipedia article***:

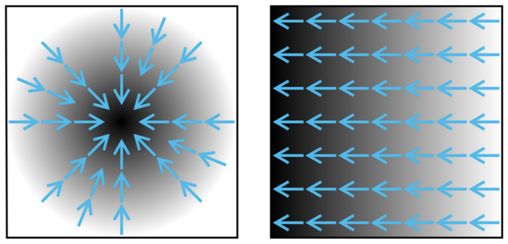

Figure 3: Gradient of two 2D functions

In this particular case, the more black, the larger the function. In the centre of the left image, the gradient is not displayed, but it would be equal to 0. Let's come back to our case. In a black and white picture, the greyscale level is analogous to the altitude, and the gradient is a measure of how fast the level changes and of the direction of the change. An edge in the picture (a black to white transition) leads to a large gradient perpendicular to the edge, from white to black. In a colour

HOG'22, December 20–24, 2022, Palma, IB, SPAIN

ISBN 978-1-4503-6708-0/20/04.
DOI: https://doi.org/10.1145/3313831.XXXXXXX

picture, one can compute a gradient for each colour level, and e.g. take the maximum gradient over the three colour levels. So why are the gradients said to be oriented? A gradient is always oriented. It just means that the method keeps track of the gradient direction, and it makes up a good acronym, HOG.

1.2 What is an histogram?

An histogram is a data structure that is used to compress data and to represent its probability distribution. An histogram can have many dimensions but in practice, 1D and 2D histograms are most often used. In HOG, the gradients are stored in a 1D histogram, so let's focus on that. Take an array of values: [1, 1.5, 2.2, 3.5, 3.5, 3.6, 4.1].
We define the histogram by describing its bins. The bins are baskets that count the number of entries with a value falling within the bin range.
You can type this in ipython to see the histogram:

```
import numpy as np
import matplotlib.pyplot as plt

values = [1.1, 1.5, 2.2, 3.5, 3.5, 3.6, 4.1]
plt.hist(values, bins=4, range=(1,5))
plt.show()
```

As shown on figure 4, the first bin has two entries (1.1, 1.5), the second bin one entry (2.2), the third bin three entries (3.5, 3.5, 3.6), and the last bin one entry (4.1)

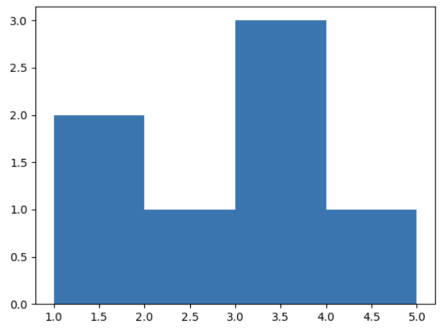

Figure 4: An histogram with 4 bins

2 Histograms of Oriented Gradients

The basic idea of the method is the following:

- The picture is scanned with a ***detection window*** of varying size.
- For each position and size of the detection window, the window is subdivided in ***cells***. The cells are in practice relatively small: they typically contain only a small part of the person to be detected, maybe the side of an arm, or the top of the head.
- In each cell, a ***gradient*** is computed for each pixel, and the gradients are used to fill an histogram: the value is the angle of the gradient, and the weight is the magnitude of the gradient.
- The histograms of all cells are put together and fed to a machine learning discriminator to decide whether the cells of the current detection window correspond to a person or not.

The last point is the subject of the next section. Here, let's see how the histogram of oriented gradients is built for a given cell, with a small example:

```
import numpy as np
import matplotlib.pyplot as plt
import cv2

cell = np.array([
[0, 1, 2, 5, 5, 5, 5, 5],
[0, 0, 1, 4, 4, 5, 5, 5],
[0, 0, 1, 3, 4, 5, 5, 5],
[0, 0, 0, 1, 2, 3, 5, 5],
[0, 0, 0, 0, 1, 2, 5, 5],
[0, 0, 0, 0, 0, 1, 3, 5],
[0, 0, 0, 0, 0, 0, 2, 5],
[0, 0, 0, 0, 0, 0, 1, 3]], dtype ='float64')

gradx= cv2.Sobel(cell,cv2.CV_64F,1,0,ksize=1)
grady= cv2.Sobel(cell,cv2.CV_64F,0,1,ksize=1)
norm,angle=cv2.cartToPolar(gradx,grady,
            angleInDegrees=True)
plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plt.imshow(cell,cmap='binary',origin='lower')
plt.subplot(1,2,2)
plt.imshow(norm,cmap='binary',origin='lower')
q = plt.quiver(gradx, grady, color='blue')
plt.savefig('gradient.png')
plt.show()
```

You get the following plots. We see that in the gradient plot, flat surfaces disappear, and the edge is clearly visible.

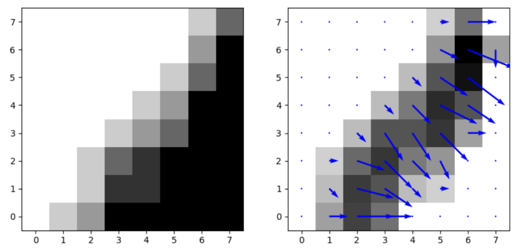

Figure 5: Left: the cell image data. Right: the corresponding gradient

The gradient is represented by an arrow, and the gradient magnitude is also shown as a greyscale color map.
Let's finally plot the histogram of oriented gradients:

```
plt.hist(angle.reshape(-1),weights=
norm.reshape(-1),bins=20,range=(0,360))
plt.show()
```

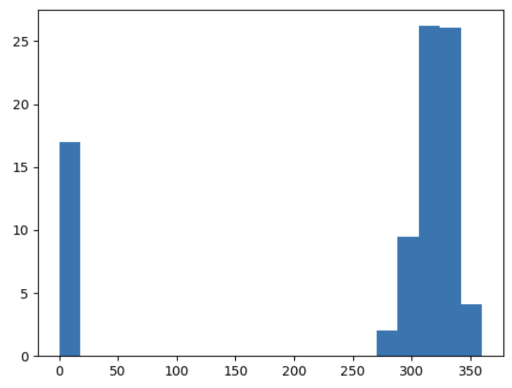

Figure 6: Histogram of gradient angles. Each entry is weighted by gradient magnitude

In this histogram, 0 corresponds to the gradients going to the right. There is only one arrow going down, shown in the bin [270;290[.
If several pixels in the cell have gradients with similar orientation, they contribute to the same bins. Also, large gradients contribute more. So The peak on the right side corresponds to the edge visible in the cell.
Histogramming the gradients in this way makes it possible to recognize edges easily in the cell.
Now this histogram is for a single cell. We need a procedure to analyze all cells in the current detection window to find out if the window contains a person or not.

3 Putting all cells together in a machine learning classifier

We have one HOG for each cell in the detection window. All HOGs are concatenated into a large array of numbers. For example, if the detection window has $8 \times 16 = 128$ cells and each HOG has 20 bins, we end up with 2560 values.
Any kind of machine learning classifier can in principle be used to take a decision (is there a person or not?) based on these values.
However, since there are a lot of dimensions (2560 in our example) and the number of images available for the training was not very large, Dalal and Triggs decided to go for a ***Support Vector Machine (SVM)*** (a linear one). That's also what is being done in the OpenCV implementation of HOG.

4 Code Implementation:

4.1 First Approche

```
import cv2
import imutils
import numpy as np

def detect(frame):
    bounding_box_cordinates,
    weights=HOGCV.detectMultiScale(frame,
    winStride = (4, 4),padding = (8, 8),
    scale = 1.03)
    person = 3
    for x,y,w,h in bounding_box_cordinates:
        cv2.rectangle(frame, (x,y),
        (x+w,y+h),(0,255,0), 2)
        cv2.putText(frame,f'person {person}',
        (x,y), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
        (0,0,255), 1)
        person += 1
    cv2.putText(frame, 'Status : Detecting ',
    (40,40), cv2.FONT_HERSHEY_DUPLEX,
    0.8, (255,0,0), 2)

    cv2.putText(frame, f'Total Persons :
    {person-1}', (40,70),
    cv2.FONT_HERSHEY_DUPLEX, 0.8,(255,0,0),2)

    return frame
```

First we implemented the function ***detect***, it takes a single argument ***frame***, which is a frame from a video stream. It uses the ***HOGCV*** object to detect people in the frame, and then it draws rectangles around them and adds text labels to the frame indicating the number of people detected. Finally, it returns the modified frame.
Note that this function relies on the ***HOGCV*** object, which has not been defined in the code you provided. You will need to initialize this object before calling the detect function.

```
def detectByPathImage(path, output_path):
    image = cv2.imread(path)
    result_image = detect(image)
    cv2.imwrite(output_path, result_image)
    return result_image

def humanDetector(image_path,output_path):
    output_result =
    detectByPathImage(image_path,output_path)

    return output_result
```

Then we implemented the ***detectByPathImage*** function takes two arguments: ***path***, which is the path to the image, and ***output_path***, which is the path where the modified image will be saved. It reads the image from the given path, detects people in it using the ***detect*** function, and then writes the modified image to the output path.
The ***humanDetector*** function is a wrapper function that calls the ***detectByPathImage*** function and returns the result. It takes two arguments: ***image_path***, which is the path to the image, and ***output_path***, which is the path where the modified image will be saved.

```
from google.colab.patches import cv2_imshow
HOGCV = cv2.HOGDescriptor()
HOGCV.setSVMDetector(
cv2.HOGDescriptor_getDefaultPeopleDetector())
output_result=
humanDetector("image01.jpg","output.png")
cv2_imshow(output_result)
```

And finally, after detecting people in the image, we show the modified image through this code.

4.2 Second Approche

```
hog = cv2.HOGDescriptor()
hog.setSVMDetector(
cv2.HOGDescriptor_getDefaultPeopleDetector())
image = cv2.imread("image01.jpg")
# keep a minimum image size for accurate
predictions
# if image width < 400
if image.shape[1] < 400:
    (height, width) = image.shape[:2]
    # find the width to height ratio
    ratio = width / float(width)
    #resize the image according to the
    width to height ratio
    image = cv2.resize(image,
    (400, width*ratio))
img_gray =cv2.cvtColor(image,
cv2.COLOR_BGR2GRAY)
rects, weights=hog.detectMultiScale(
img_gray, winStride=(2, 2),
padding=(10, 10), scale=1.02)
for i, (x, y, w, h) in enumerate(rects):
    if weights[i] < 0.13:
        continue
    elif weights[i]<0.3 and weights[i]>0.13:
      cv2.rectangle(image, (x, y),
        (x+w, y+h), (0, 0, 255), 2)
    if weights[i] < 0.7 and weights[i]>0.3:
      cv2.rectangle(image, (x, y),(x+w, y+h),
        (50, 122, 255), 2)
    if weights[i] > 0.7:
      cv2.rectangle(image, (x, y),(x+w, y+h),
        (0, 255, 0), 2)
cv2.putText(image, 'High confidence',
(10, 15), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(0, 255, 0), 2)
cv2.putText(image, 'Moderate confidence',
(10, 35), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(50, 122, 255), 2)
cv2.putText(image, 'Low confidence',
(10, 55), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(0, 0, 255), 2)
cv2_imshow(image)
#cv2.imshow('HOG detection', image)
cv2.imwrite("new_algo.jpg", image)
#cv2.waitKey(0)
```

This code initializes the ***HOG descriptor/person detector*** using the default people detector provided by ***OpenCV***. It reads an image and resizes it if its width is less than 400 pixels. It then converts the image to ***grayscale*** and uses the ***HOG*** descriptor to detect people in the image. Finally, it draws rectangles around the detected people, using different colors and adds text to the image to indicateand the confidence level of the detection. It then shows the image and writes it to a file.

5 Code Performance:

The code uses the HOG descriptor and SVM classifier to detect people in an image. The performance of this person detection task will depend on a number of factors, such as the size and complexity of the input image, the efficiency of the HOG descriptor and SVM classifier, and the hardware resources available.
To calculate the size and complexity of the input image, you can use the ***shape*** attribute of the image array, which returns a tuple with the number of rows, columns, and channels in the image.

```
import cv2

# read the image
image = cv2.imread("image01.jpg")
# calculate the size and complexity
   of the image
rows, columns, channels = image.shape
size = rows * columns
complexity = size * channels
print(f"Size: {size} pixels")
print(f"Complexity: {complexity} pixels")
```

To calculate the efficiency of the HOG descriptor, you can measure the time it takes to compute the HOG descriptor for the image. You can use the time module to measure the elapsed time.

```
import cv2
import time

# initialize the HOG descriptor
hog = cv2.HOGDescriptor()
# read the image
image = cv2.imread("image01.jpg")
# convert the image to grayscale
img_gray = cv2.cvtColor(image,
            cv2.COLOR_BGR2GRAY)
# measure the time it takes to
  compute the HOG descriptor
start = time.time()
hog_descriptor = hog.compute(img_gray)
end = time.time()
# calculate the efficiency of
   the HOG descriptor
efficiency = 1.0 / (end - start)
print(f"Efficiency: {efficiency} HOG/s")
```

We did not get the desired results from the program, because he detect a few persons from our picture. But we can improve the performance of this code, we can try the following:

- Use a more efficient implementation of the HOG descriptor and SVM classifier, such as the one provided by OpenCV.
- Optimize the parameters of the HOG descriptor and SVM classifier, such as the window size, stride, padding, and scale, to achieve the best trade-off between accuracy and speed.