An Introduction into Computer Vision: The Histogram of Oriented Gradients

CIS 492 Computer Science Senior Seminar

Matthew Clark

May 8, 2018

Abstract

In recent years, due to significant improvements of sensors, processors, and image processing techniques, major milestones have been made in the field of autonomous vehicles. As a result, the automotive industry and consumers have developed a deep interest in the substantial socioeconomic impact autonomous vehicles have. To realize full autonomous driving, the vehicle must have reliable object detection systems. Lidar, cameras, and other proximity sensors are used to make this possible. Much of the current research and development is being poured into improving various object recognition techniques. A proper background and understanding of Computer Vision is vital to ensure advancement in applications like autonomous vehicles. To provide an effective tool to begin to understand the state of the field in its current form, this paper is devoted to the survey and analysis of the Histogram of Oriented Gradients technique.

Contents

1 Introduction

Computer vision is a vast and deep field within computer science with little recognition and yet widespread impact. Reflecting upon the significant milestones that represent the advancements of intelligent computing, a computer beating a chess grandmaster seems more *intelligent* than a computer learning to recognize a face in an image. In reality, barely recognizing a face in an image had it's first successes in the early 90's whereas a computer chess system was able to find success in 1970 against a chess grandmaster and was capable of beating people since the late 1950's [1]. In the 1970's it was believed that the "visual input" problem for the ambitious goal of mimicking human intelligence was an easy task compared to problems involving higher-level reasoning and planning [2]. Those early pioneers soon realized the immense complexity and difficulty of computer vision.

Formally, the goal of computer vision is to teach computers how to see [3][4]. Studying computer vision, a field of applied math, requires geometry, calculus, linear algebra, probability, and machine learning. Despite the very narrow entryway, it is both academically enriching and a specialty that is extremely high in demand. Within the field of computer vision, object recognition is a relevant and integral application for autonomous vehicles. Although other applications of object recognition can be applied to surveillance systems, facial recognition, or intelligent photo editing. Within an autonomous vehicle system, object recognition is best suited for various tasks such as recognizing road lanes, surrounding vehicles, hazards, road signs, pedestrians, etc. To understand object recognition, this paper is devoted to providing a comprehensive review of the Histograms of Oriented Gradients technique[1] for Pedestrian Detection [5], describing the necessary and foundational concepts.

2 Image Processing

Before we are able to understand the HOG [5] technique, we need to review the topic of image processing. **Preprocessing** is the use of image processing to convert the input, an image, into a more suitable form for later analysis [2]. Generally the goal

[1] From now on we will refer to the Histograms of Oriented Gradients as HOG

of preprocessing is to remove unnecessary data, correct the image, or alter its representation. Many of these tasks are computed using point operators, histograms, filters, gradients, or image pyramids. As a matter of fact, all of these tasks are necessary when using HOG.

(a) The Input Image (b) The Output Image

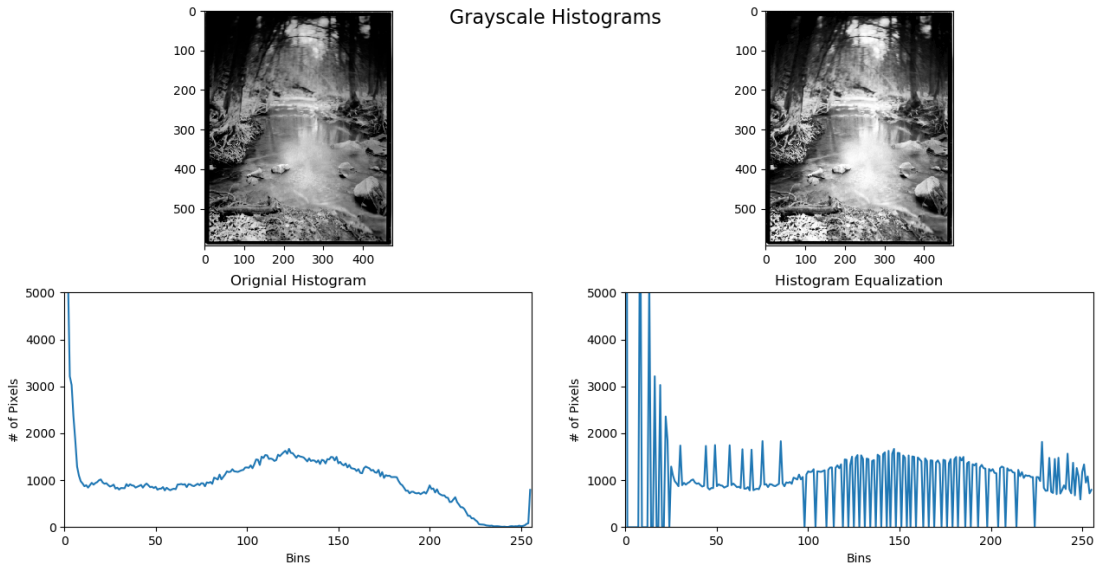
Figure 1: Examples a representation of a simple point operation where $O_{ij} = (I_{ij}) + 20$

2.1 Point Operators

Typically the simplest operation in image processing, point operators, deal with pixel transformations also known as a **point process**. The output image at each pixel depends only on the corresponding input pixel data. This means that, for an input image I and output image O of size $m \times n$ where I_{ij} is a pixel location,

$$O_{ij} = a(I_{ij}) + b \tag{2.1}$$

represents a general form for a pixel transformation. This assumes that $a > 0$ and $\forall a, b \in \mathbb{R}$. The value a is called the *gain* and b is *bias*. In other words, to increase brightness, one would increase the *bias* and to increase the contrast (see Fig. 1), one would increase the *gain*.



(a) The Input Image

Figure 2: Histogram Equalization produces a far smoother contrast effect compared to Fig. 1 above.

2.2 Histograms

Suppose that we want to visualize the changes in our data or represent it in a more compact fashion, forming a distribution like a histogram is a powerful tool to accomplish this. The idea of a distribution is essentially representing how the color values are distributed across an image. For instance, at the pixel value 200 red there might be 1000 pixels containing that color data. Histograms also allow for more dynamic transformations of various pixel data. This means that instead of increasing our contrast with the point function, we will use a histogram of pixel values ranging from [0 - 256] (RGB) for each color channel (see Fig. 2) to transform our input. This is done using the form of the general **image processing operator**:

$$g(x,y) = h(f(x,y)) \qquad \text{or} \qquad g(x,y) = h(f_0(x,y), \ldots, f_n(x,y)) \tag{2.2}$$

Notice in Figure 1, there are a large amount of artifacts [6] that are produced by our contrast point operation (2.1). To make our contrast operation smoother, we will sum the histograms in the range $0 \leq I$ for each I^{th} bin of 256 bins. Imagine calculating the grades of a 500 student class; it is much simpler to represent the percent valued grades of each student as a letter grade associated with a group of students. Our "grades" are then used to transform and describe our original image. This will

form a **cumulative distribution** $c(I)$ from the histograms $h(I)$ that will be rescaled and mapped back to the image. Equation 2.3 is the contrast operation and equation 2.4 is the mapping function, also known as a linear blend function where $f(I)$ is our output image.

$$c(I) = \frac{1}{N}\sum_{j=0}^{I} h(i) = c(I-1) + \frac{1}{N}h(I) \tag{2.3}$$

$$f(I) = \sigma c(I) + (1-\sigma) \tag{2.4}$$

This means that applying gain to each pixel intensity is more spatially dynamic and non-linear.

Transitioning from using a contrast point function to cumulative distribution is similar to moving from graphing a linear equation like $f(x) = 2x + 1$ to graphing $f(x) = \int(\sum_{k=0}^{x} ak + b)dx)$. Formally this process is known as a **histogram equalization** or CLASHE [7], and we see later that gradients involve a similar process. A variant of histogram equalization is used in a popular object recognition technique called SIFT [8, Lowe 2004] or Scale Invariant Feature Transforms.

2.3 Filters

Instead of using *global* image information such as a histogram of the color channel distribution, the goal of image filters is to apply some operation to all or some of the pixel values in an image. For example, image filters can be weighted combinations of pixels in small neighborhoods. All image filters include some form of **kernel operation**. To understand what kernels are there a few fundamental linear algebra concepts that need to be formed.

Linear Systems

Matrices can also be a form of a linear system, $a_1x_1 + a_2x_2 + ... + a_nx_n$ that allows for more efficient solutions. For example, provided the following linear system, find all

the variables that solve the equation correctly:

$$x + 2y + 3z = 6$$
$$2x - 3y + 2z = 14$$
$$3x + y - z = -2$$

It is possible to solve this equation with the *method of elimination*, that is eliminating some variables by adding one equation to another. There is a better way to solve this problem, by representing the equation as a matrix:

$$A_{coefficients} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & -3 & 2 \\ 3 & 1 & -1 \end{bmatrix} \quad x_{variables} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad b_{solution} = \begin{bmatrix} 6 \\ 14 \\ -2 \end{bmatrix} \tag{2.5}$$

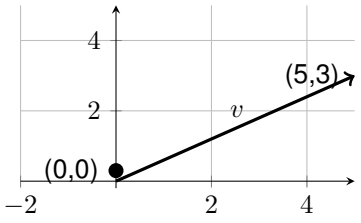
The goal remains the same, find the value of one of the variables. Instead of using the *method of elimination*, we will transform the coefficient matrix A into a Row Echelon Form (See the Wolfram Article):

$$REF\left(\left[\begin{array}{ccc|c} 1 & 2 & 3 & 6 \\ 2 & -3 & 2 & 14 \\ 3 & 1 & -1 & -2 \end{array}\right]\right) = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & -2 \\ 0 & 0 & 1 & 3 \end{array}\right] = \begin{array}{rcl} 1x + 0y + 0z & = & 1 \\ 0x + 1y + 0z & = & -2 \\ 0x + 0y + 1z & = & 3 \end{array} \tag{2.6}$$

As shown above, it is possible to solve our linear system using a Row Echelon matrix. Later this matrix transformation will be used to help compute a kernel.

Matrix Transformations

A Matrix in its most basic form is a rectangular array of size mn with m horizontal rows and n vertical columns. Likewise an image can be thought of as a pixel matrix of size mn, and typically a pixel can also be though of as a matrix of size $1 \times n$ since it typically contains some three-channel 8-bit color data. A matrix of size $1 \times n$ or $n \times 1$ is called a vector and can be used to find magnitude and direction. Vectors are used in various applications such as multi-variable calculus or linear algebra. The following are two good visualizations of a vector.



$$v = \begin{bmatrix} 5 \\ 3 \end{bmatrix} \tag{2.7}$$

Matrices and vectors are not simply a different notation of representing numbers, but contain powerful and unique properties and operations. For example, the traveling salesman problem can be solved by the Hungarian Method in $O(n^3)$ time [9]. This is done using a matrix that represents the costs and performs various matrix transformations to reliably solve the problem. Scalar multiplication and addition, transposition, and inverses are the basic matrix transformations and building blocks for geometric transformations. Finding the direction and magnitude of a vector is vital to understanding HOG and other feature descriptors.

A **scalar** is a single real number that is scaled to appropriate units of measure [10]. In other words, a scalar is used to resize an image or change the pixel values of the overall image like in the contrast point process (2.1). Applying a scalar to a matrix would operate as follows:

$$3\begin{bmatrix}2 & 1\\ 5 & 3\end{bmatrix} = \begin{bmatrix}6 & 3\\ 15 & 9\end{bmatrix} \tag{2.8}$$

Following scalars, transposing a matrix is a powerful tool for transforming matrices. Denoted by the symbol A^T where A is any matrix, the **transpose** of the matrix will interchange the rows and columns of A, implying that $a_{ij}^T = a_{ji}$.

$$A = \begin{bmatrix}a_{11} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}\end{bmatrix}, A^T = \begin{bmatrix}a_{11} & a_{21}\\ a_{12} & a_{22}\\ a_{13} & a_{23}\end{bmatrix} \tag{2.9}$$

Instead of multiplying a single value to a matrix, it also possible to multiply matrices together, also known as a **matrix multiplication**. This is done by multiplying two matrices, $A = [a_{ij}]$ of size $m \times p$ and $B = [b_{ij}]^T$ of size $p \times n$.

$$A = \begin{bmatrix}1 & 2 & -1\\ 3 & 1 & 4\end{bmatrix},\ B = \begin{bmatrix}-2 & 5\\ 4 & -3\\ 2 & 1\end{bmatrix}$$

$$\sum_{k=1}^{p} a_{ik}a_{jk},\ A \times B = \begin{bmatrix} (1)(-2)+(2)(4)+(-1)(2) & (1)(5)+(2)(-3)+(-1)(1) \\ (3)(-2)+(1)(4)+(4)(2) & (3)(5)+(1)(-3)+(4)(1) \end{bmatrix}$$
$$= \begin{bmatrix} 4 & -2 \\ 6 & 16 \end{bmatrix} \tag{2.10}$$

The **inverse** of a matrix is a powerful tool for forming a wide variety of both color and geometric changes. A matrix A of size $n \times n$ is invertible if there is some matrix A^{-1} that $AA^{-1} = I_n$. The matrix I_n is an identity matrix, which is a matrix of 1's on the diagonal $I_{ii}, (0 \leq i \leq n)$ and 0's everywhere else.

$$I_n \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} \tag{2.11}$$

Therefore the inverse would be found by obtaining a linear system and finding values that will satisfy our definition of an inverse.

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \qquad A^{-1} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \tag{2.12}$$

Solve

$$AA^{-1} = \begin{bmatrix} a+2c & b+2d \\ 3a+4c & 3b+4d \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{2.13}$$

$$A^{-1} = \begin{bmatrix} -2 & 1 \\ \frac{3}{2} & -\frac{1}{2} \end{bmatrix}$$

Suppose that our vector **v** with two elements represents the x and y points on a graph. How would we find the length of **v** in the graph (2.7)? Simple, since the x and y elements of our vector form a triangle, use Pythagorean theorem to find it:

$$\|\mathbf{v}\| = \sqrt{v_x^2 + v_y^2} \tag{2.14}$$

In computer vision this is called the norm or magnitude of a vector, since the length of **v** is also the measure of change from the origin to the point. Since there are two points to measure, the y-axis ($\mathbf{u} = [0, y]$) and the vector elements ($\mathbf{v} = [5, 3]$), it is possible to

find the direction of line:

$$cos\theta = \frac{u_1 v_1 + u_2 v_2}{\|\mathbf{u}\| \, \|\mathbf{v}\|} \tag{2.15}$$

In this case it our $\|\mathbf{v}\|$ is 5.83... and the direction is 49θ. The value $y \neq 0$ can be any real number and cannot impact the direction formula. Our norm can be used to convert the values of **v** to be in the range [0-1].

Geometric Transformations

Using the concepts of matrix transformations, it is possible to alter the geometric properties of a matrix. In preprocessing, it is used to allow for more robust feature matching. SIFT or Scale Invariant Feature Transforms[8] produce features that are geometrically invariant of affine changes. Before the term affine is defined; reflections, rotations, shears, scale, and translations must be reviewed.

The Reflection of a matrix with respect to the x-axis or y-axis is a matrix-vector product of an diagonal matrix and a n-dimensional point. In matrix multiplication, the second matrix was transposed to multiply each element row by row. With the matrix-vector product, it will produce a vector that has been *scaled* by a vector with the initial matrix. In the case of reflection, the following matrices will rotate vectors' values with respect to the x or y axis:

$$A = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \mathbf{v}_{x-axis} = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \\ -y \end{bmatrix}, \qquad \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \left[\begin{array}{c|c|c} -1 & 3 & 2 \\ 4 & 1 & 6 \end{array}\right] = \left[\begin{array}{c|c|c} -1 & 3 & 2 \\ -4 & -1 & -6 \end{array}\right] \tag{2.16}$$

$$A = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \mathbf{v}_{y-axis} = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -y \\ -x \end{bmatrix}, \qquad \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \left[\begin{array}{c|c|c} -1 & 3 & 2 \\ 4 & 1 & 6 \end{array}\right] = \left[\begin{array}{c|c|c} -4 & -1 & -6 \\ 1 & -3 & -2 \end{array}\right] \tag{2.17}$$

Rotations are also a matrix-vector product with one important difference, the matrix is a composed of sines and cosines with an arbitrary angle θ.

$$A\mathbf{v} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \left[\begin{array}{c|c|c} -1 & 3 & 2 \\ 4 & 1 & 6 \end{array}\right] = \left[\begin{array}{c|c|c} -4 & -1 & -6 \\ -1 & 3 & 2 \end{array}\right] \tag{2.18}$$

Unlike a reflection or rotation, which maintain the *shape* of the object formed by the points, shears allow for the transformation of one direction. This is quite useful for correcting misaligned objects within an image, which can dramatically decrease the FPPW (False Positives per Window) [cite example]. The scalar k can be any real number, and tends to be ether the $cot\theta$ or some constant. In this case, $k = cot(10)$:

$$A\mathbf{v} = \begin{bmatrix} 1 & k \\ 0 & 1 \end{bmatrix} \left[\begin{array}{c|c|c} -1 & 3 & 2 \\ 4 & 1 & 6 \end{array}\right] = \left[\begin{array}{c|c|c} 5.2 & 4.5 & 11.3 \\ 4 & 3 & 2 \end{array}\right] \tag{2.19}$$

Notice how the second row, the y-axis, was unaffected by the shear operator.

Scaling a matrix is one of the simplest geometric operations, it essentially applies a scalar to the x and y values respectively. Therefore, if we wanted to scale our triangle to be double its size, simply apply the following formula with $\beta = [2, 2]$:

$$A\mathbf{v} = \begin{bmatrix} \beta_x & 0 \\ 0 & \beta_x \end{bmatrix} \left[\begin{array}{c|c|c} -1 & 3 & 2 \\ 4 & 2 & 6 \end{array}\right] = \left[\begin{array}{c|c|c} -2 & 6 & 4 \\ 8 & 2 & 12 \end{array}\right] \tag{2.20}$$

Up to this point, all of the transformations have been scalar based, but it is also possible to perform vector-addition to move the triangle around the plane. Suppose that we wanted to displace or move the points of the triangle to the right, we simply add some constant $\sigma = [2, 0]$ to each element in the point.

$$T = \begin{bmatrix} \sigma_x \\ \sigma_y \end{bmatrix} + \left[\begin{array}{c|c|c} -1 & 3 & 2 \\ 4 & 2 & 6 \end{array}\right] = \left[\begin{array}{c|c|c} 1 & 5 & 4 \\ 4 & 2 & 6 \end{array}\right] \tag{2.21}$$

Notice that all of these transformations are dealing with x and y points on a Cartesian Plane. This means that all of the transformations are spatial or in space. This is exactly what the term affine defines, it is an extension of a vector space that includes an additional type of object, the point. Therefore, when papers like [Distinctive Image Features from Scale-Invariant Keypoints 8] describes its features as scale and affine invariant, it means that the features do not vary spatially. This is why SIFT was often used in moving scenes for large objects.

Vector Spaces

Another integral component to a kernel is a vector space, which are defined as a set of objects and two operations $\oplus$ and $\odot$ on these objects (See the Wolfram Article). The set of objects in this case is a set of vectors denoted as R^n[1] with each vector of size n. Now to understand what the operations $\oplus$ and $\odot$ do starting with $\oplus$, suppose we have two $1 \times n$ matrices where $\oplus$ is denoted as a sum operator or **vector addition**:

$$\begin{bmatrix} a_1 & a_2 & \dots & a_n \end{bmatrix} \oplus \begin{bmatrix} b_1 & b_2 & \dots & b_n \end{bmatrix} = \begin{bmatrix} a_1 + b_1 & a_2 + b_2 & \dots & a_n + b_n \end{bmatrix} \tag{2.22}$$

The multiplication operator $\odot$ or **scalar multiplication** works the same way:

$$A \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \odot B \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & a_{13}b_{13} \\ a_{21}b_{21} & a_{22}b_{22} & a_{13}b_{13} \\ a_{31}b_{31} & a_{32}b_{32} & a_{33}b_{33} \end{bmatrix} \tag{2.23}$$

The resulting matrix would thus be denoted as a vector space of M_{33}. Using the properties of a matrix and vectors and operations of a vector space, we can now perform a kernel operation. We want to sum the scalar multiplier $(A \odot C)$ and plug in values some pixel values into A and an edge detection kernel into B.

$$A = \begin{bmatrix} 158 & 249 & 125 \\ 211 & 227 & 114 \\ 178 & 246 & 123 \end{bmatrix} \text{ and } B = \begin{bmatrix} -1 & -1 & 1 \\ -1 & 8 & 1 \\ -1 & -1 & 1 \end{bmatrix} \tag{2.24}$$

C is defined as $B = R(R(B)^T)^T$. This produces the effect of swapping the rows and columns by rotating the transposition of the rotation of B^T, which looks like so:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} a_{33} & a_{32} & a_{31} \\ a_{23} & a_{22} & a_{21} \\ a_{13} & a_{12} & a_{11} \end{bmatrix} \tag{2.25}$$

[1] R^n is the set of $n \times 1$, column matrices, R_n is the set of $n \times 1$, row matrices, and M^{mn} is set of $m \times n$ matrices

Now perform the kernel operation:

$$\begin{aligned}\sum A \odot C &= (a_{11}b_{33}) + (a_{12}b_{32}) + (a_{13}b_{31}) + (a_{21}b_{23}) + (a_{22}b_{22}) + (a_{13}b_{21}) \\ &+ (a_{31}b_{13}) + (a_{32}b_{12}) + (a_{33}b_{11}) \\ &= (158 \cdot -1) + (249 \cdot -1) + (125 \cdot -1) + (211 \cdot -1) + (227 \cdot 8) + (114 \cdot -1) \\ &+ (178 \cdot -1) + (246 \cdot -1) + (123 \cdot -1) \\ &= 412 \end{aligned} \tag{2.26}$$

There is a problem, our output is too large to be considered a pixel value and therefore we must normalize by dividing the output by the sum of our kernel matrix. This operation of summing the scalar multiplication of two vector spaces is more widely known as a **kernel convolution** (denoted by the symbol $*$ or $\otimes$). The resulting value that has been normalized is the value of the center pixel, a_{22}. When implemented, a kernel convolution follows the following formula, where s is the size of the kernel matrix:

$$O[i,j] = \sum_{l=-s}^{s} \sum_{k=-s}^{s} I[u,v]K[i-l, j-k] \tag{2.27}$$

Sometimes it is easier to understand a convolution by its discrete operator above or its matrix operator in the equations 2.24, 2.25, (2.26). Another type of kernel operation that is also often used in image processing is called **Cross-Correlation** (also denoted by the symbol $\otimes$ - pay attention to the discrete form) and is simply the sum of the scalar multiplier ($A \odot B$) where B is not flipped. In discrete form it looks like so:

$$O[i,j] = \sum_{l=-s}^{s} \sum_{k=-s}^{s} I[u,v]K[i+l, j+k] \tag{2.28}$$

2.4 Gradients

Unlike a simple edge detection kernel shown in equation 2.27, gradients allow for more complex kernel operations that are formed by a non-linear equation. Our gradient equation will take in a pixel value at i and j for all pixels in an image and produce an image that represents the change of our image with respect to both x and y of the image function $f(x,y)$. The formal definition of a gradient is a partial derivative of a vector-valued function of two variables:

$$\nabla F(x,y) = f_x(x,y)\mathbf{i} + f_y(x,y)\mathbf{j} \tag{2.29}$$

$$\nabla F = \begin{bmatrix} \frac{\partial F}{\partial x} & \frac{\partial F}{\partial y} \end{bmatrix} \tag{2.30}$$

The two equations represent the same gradient operation in two forms. The first is in the form of a vector-valued function[1] that will produce the gradient vector $\begin{bmatrix} f_x(x,y) & f_y(x,y) \end{bmatrix}$. Therefore, to be able to understand and compute this formula, it is necessary to briefly introduce the concept of partial derivatives.

Partial Derivatives

In Calculus, the finding a derivative is in essence finding the tangent of a line. A tangent is simply the slope or rate of change at a point in space for some function. An affine vector is also the measure of direction (2.15) and magnitude (2.14) in space that is transformed by some system of linear equations. Finding the slope for all the linear equations formed by the affine matrix is also known as a **partial derivative**. This means that for the function $f(x,y) = x^2y + sin(y)$ at the point $(1,2)$,

$$\frac{\partial f}{\partial x}(x,y) \Longrightarrow \frac{\partial f}{\partial x}(x^2y + sin(y)) \Longrightarrow f_x(x,y) = 2xy + 0$$
$$f_x(x,y) = 2xy + 0$$
$$f_x(1,2) = 4$$

$$\frac{\partial f}{\partial y}(x,y) \Longrightarrow \frac{\partial f}{\partial y}(1^2y + cos(y)) \Longrightarrow f_y(x,y) = 1 + cos(y)$$
$$f_y(x,y) = x^2 + cos(y)$$
$$f_y(1,2) = 0.5838\ldots \tag{2.31}$$

Now that we have found the partial derivatives, we can put $f_x(1,2)$ and $f_y(1,2)$ back into the form of equation 2.29:

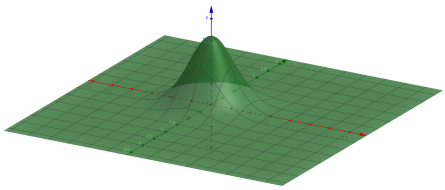
$$\nabla f(1,2) = 4\mathbf{i} + 0.5838\mathbf{j} \tag{2.32}$$

Calculation

Now that we are able to find the partial derivative, it is possible to form a gradient image. A popular gradient function

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.33}$$

[1] The concept of a vector-valued function is functionally the same as an affine transformation.



(a) 2-D Distribution

$\frac{1}{273}$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

(b) Gausian Kernel

Figure 3: Example of a 2-D Gradient Distribution that forms a Gaussian Convolution Kernel, 273 is the sum of all the values in the kernel filter

where σ is some constant smooths the image. Since our gradient produces a vector at each pixel, it can be used to also form a convolution kernel with some predetermined values. The convolution kernel produced by our gradient function is also known as Gaussian Smoothing see Fig. 3. If a white image was given as input, there would be no change at any point in our image and thus the gradient vector for every point or pixel would look like so:

$$\nabla F = \begin{bmatrix} 0 & 0 \end{bmatrix} \tag{2.34}$$

If there was change only in the x or y direction,

$$\nabla F_x = \begin{bmatrix} \frac{\partial F}{\partial x} & 0 \end{bmatrix}, \quad \nabla F_y = \begin{bmatrix} 0 & \frac{\partial F}{\partial y} \end{bmatrix} \tag{2.35}$$

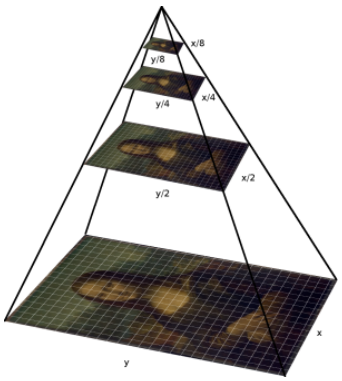
Once the gradient has been computed for some pixel ij in the image, it is possible to find the direction (2.36) and magnitude (2.37) of the gradient vector using the following formulas:

$$\theta = tan^{-1}\left(\frac{\frac{\partial F}{\partial x}}{\frac{\partial F}{\partial y}}\right) \tag{2.36}$$

$$\|\nabla F\| = \sqrt{(\frac{\partial F}{\partial x})^2 + (\frac{\partial F}{\partial y})^2} \tag{2.37}$$

2.5 Image Pyramids

The final key concept necessary to understanding HOG are **image pyramids**. They are a multi-resolution representation of an image and are used to rescale the image to various sizes for detecting different objects at different scales. Image pyramids find objects at various locations by using a **sliding window**, which is also called the **detection window**. Calculating an image pyramid of n levels involves two processes, Gaussian smoothing which was described



(a) Image Pyramid

Figure 4: The traditional image pyramid, where the base is most fine resolution and the top is the most corse. Each level is half the size of the previous.

above (see 3) and sampling. At each level, we just store the differences between the image at that level and the predicted image from the next level.

The concept of sampling is derived from Wavelets and Fourier Transforms. **Sampling** is defined as the process of producing a finite set of points from a continuous wavelet that has been derived from an image. Similar to the cumulative distribution in section 2.2, wavelets and Fourier transforms are ordinary multivalued functions of one dimension applied to signals. Wavelets and Fourier transforms have a number of applications, such as audio, signal or image processing. How could an image be a signal? Consider the fact that an image is a collection of two dimensional data points that can be graphed to a Cartesian Plane. Audio signals are the same, yet are a collection of one dimensional data points. As the concept of partial derivatives has shown, x and y can be separated to produce two functions accordingly. Therefore an image is a signal or two signals with the functions $f(x)$ and of $f(y)$.

Fourier transforms are only sinusoid or sin wave functions, whereas wavelets have a wide variety of families. Wavelets are also localized in space and therefore are variable at different window sizes; however, Fourier transforms are affine invariant. This implies that wavelets are better suited for recognizing spatially bound objects. Many of the earliest real-time object detectors utilized a special type of wavelet known as a Haar wavelet [11]. This framework was later improved upon in SURF or Speeded-Up Robust Features [12].

When creating an image pyramid, sampling is used to produce an image of either higher or lower[1] resolution, hence the name**multi-resolution**. The two types of sampling are **interpolation** or *upsampling*, and **decimation** or *downsampling*.

[1]Oftentimes a higher resolution or size of an object such as an image or histogram is referred to as fine, and inversely coarse for a lower resolution.

To sample an image, one must use a wavelet or Fourier transform to produce a convolution kernel that will be plugged into one of the two formulas, either for upsampling (2.38) or downsampling (2.39).

$$O[i,j] = \sum_{l=-s}^{s} \sum_{k=-s}^{s} I[u,v]K[i-rl, j-rk] \tag{2.38}$$

$$O[i,j] = \sum_{l=-s}^{s} \sum_{k=-s}^{s} I[u,v]K[ri-l, rj-k] \tag{2.39}$$

The constant r is the sampling rate. In upsampling it is used to make the kernel deal with a finer scale of pixels, whereas, in downsampling it reduces the scale of the pixels.

3 HOG Feature Descriptors

Feature Descriptors reflect a major component within the field of computer vision. Various types of feature descriptors are used to analyze, describe, and match images. There are two general types of features, key-point or appearance profiles and edge profiles, and both are some form of affine vector space object. HOG's features are edge based, which are simpler and easier to understand compared to key-point descriptors. There are five general steps to creating HOG features, aimed at evaluating well-normalized local histograms of image gradient orientations in a dense grid.

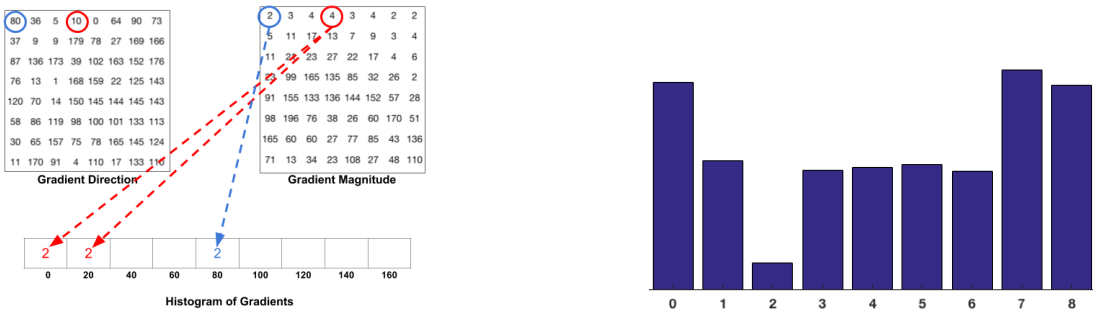
3.1 Preprocessing

The first step in the HOG process is to prepare the images to produce the most robust feature set possible. Since there is a lot of unnecessary information and our objects within the image have a unique scale, it is important to filter and form an image pyramid. Typically the initial filters applied to the image are normalized corrections of gamma[1] and color. Since the location of the objects is unknown, it is necessary to use a detection window scaled to each level in the pyramid. This window will *crop* the image and describe the features within that sub-image only. The pyramids will then be calculated using Gaussian Smoothing and then downsampled for a number of scales, typically 4 or 8 scales are used.

3.2 Gradient Images

Once the image has been preprocessed, the gradients can be calculated. There are various type of gradient filters available to use, but for simplicity sake, a 1-D centered filter for

[1] Also known as exposure or overall intensity of an image.



(a) Oritation Bins (b) 8x8 Histogram

Figure 5: The 9×1 histograms formed by our gradient vectors for a descriptor block [26].

both x or y.

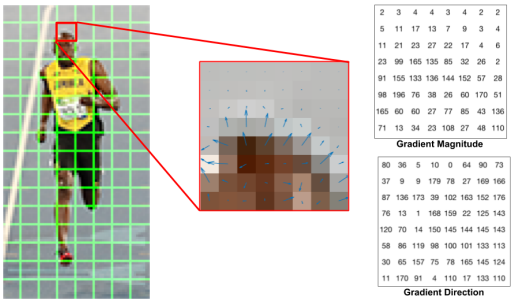
$$\begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \tag{3.1}$$

Two images will be produced, one for the x-gradient image and the y-gradient image. From these two images, the magnitude and direction can be computed and is now ready to be placed in histograms. Not only does the gradient calculation provide the essential data for the features, it will also remove a large amount of unnecessary data.

3.3 Gradient Histograms

Now that the gradients have been prepared, it is time to place them into histograms. Unlike the histograms of color data in section 2.2, the histogram will be formed with 9 orientation bins of angles $0 - 180^o$ as the x-axis and the sum of the gradient magnitudes as the y-axis. A gradient's direction will denote what direction bin the magnitude will be added to. Notice figure 5, the 9 bins are given a direction, if the gradient direction is split between two bins the magnitude will be evenly divided among the two bins.

Since each histogram describes a single edge orientation and magnitude, there must be multiple smaller histograms to provide accurate edge features. For an object is made up of various edge orientations, not simply a single edge feature. Therefore the image will split up into blocks, which can be used in various sizes dependent on the object scale. In a 64×128 size detection window, 8×8 blocks are recommended. Inside each block a gradient histogram is calculated.



(a) HOG Gradients

(b) HOG Blocks

Figure 6: The (a) descriptor blocks formed by the gradient histograms and (b) the devision of descriptor blocks. [26]

3.4 Block Normalization

Once the gradient histograms have been produced, the magnitudes within each histogram needs to be normalized to its neighboring blocks. This allows for the features to be more invariant to dramatic pixel changes or irregularities. Instead of normalizing the entire 64×128 size detection window, 4x4 blocks will be computed. These larger blocks concatenate the 9×1 histogram vectors to a 36×1 vector. The 4x4 norm block will traverse through the entire image by only moving one block horizontally or vertically at a time, see Figure 6

3.5 Finalize - Support Vector Machines

The blocks have been normalized, the 36×1 vectors will be concatenated into one large 3780×1 vector that is sent off to a Support Vector Machine. A Support Vector Machine or SVM is a machine learning technique that binaurally classifies objects. In other words, given some data set of only positive objects or training set and an input of new objects, a SVM will provide the set of new objects that are classified as matching the training set. In the case of HOG, the objects will be the HOG features of each image.

4 Further Research

The topics and concepts covered in this paper are merely the tip of the iceberg, and it is highly recommended that further study be done in the fields of applied math. A number of textbooks were used in the writing of the paper, and would be helpful resources. The two most significant are [Computer Vision: Algorithms and Application 2] and [Elementary Linear Algebra

with Applications 13]. Linear Algebra is the base for many of the concepts within computer vision and ought to be further reviewed. Szeliski's book on computer vision is one of the most well cited pieces of literature within the field of computer vision. It is a book that ought to be read after some of the *mathematical groundwork* has been laid.

References

[1] W. Contributors, *Computer chess - Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Computer_chess

[2] R. Szeliski, *Computer Vision: Algorithms and Applications*, 11th ed., ser. Texts in Computer Science. Springer London, vol. 5. [Online]. Available: http://research.microsoft.com/en-us/um/people/szeliski/book/drafts/szelski_20080330am_draft.pdfhttp://link.springer.com/10.1007/978-1-84882-935-0

[3] Nat and Friends, "(36) how computer vision is finally taking off, after 50 years - YouTube," https://www.youtube.com/watch?v=eQLcDmfmGB0. [Online]. Available: https://www.youtube.com/watch?v=eQLcDmfmGB0

[4] J. Malik, P. Arbeláez, J. Carreira, K. Fragkiadaki, R. Girshick, G. Gkioxari, S. Gupta, B. Hariharan, A. Kar, and S. Tulsiani, "The three r's of computer vision: Recognition, reconstruction and reorganization," vol. 72, pp. 4–14.

[5] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1. IEEE, pp. 886–893. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=1467360&contentType=Conference+Publications&searchField=Search_All&queryText=Histograms+of+oriented+gradients+for+human+detectionhttp://ieeexplore.ieee.org/document/1467360/

[6] W. contributors, *Distortion (optics)*. [Online]. Available: https://en.wikipedia.org/wiki/Distortion_(optics)

[7] S. M. Pizer, E. P. Amburn, J. D. Austin, R. Cromartie, A. Geselowitz, T. Greer, B. ter Haar Romeny, J. B. Zimmerman, and K. Zuiderveld, "Adaptive histogram equalization and its variations," vol. 39, no. 3, pp. 355–368.

[8] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," vol. 60, no. 2, pp. 91–110.

[9] Baripada College, J. Nayak, and S. Nanda, "Hungarian method to solve travelling salesman problem with fuzzy cost," vol. 49, no. 5, pp. 281–284. [Online]. Available: http://www.ijmttjournal.org/archive/ijmtt-v49p544

[10] R. Larson and B. H. Edwards, *Calculus*, 10th ed. Brooks/Cole, Cengage Learning.

[11] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," vol. 1. IEEE Comput. Soc, pp. I–511–I–518. [Online]. Available: http://ieeexplore.ieee.org/document/990517/

[12] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (SURF)," vol. 110, no. 3, pp. 346–359.

[13] B. Kolman and D. R. Hill, *Elementary linear algebra with applications*, 9th ed. Pearson Prentice Hall, OCLC: ocn148033544.

[14] J. C. Thomas, M. Sage, J. Dillenberg, and V. J. Guillory, "A code of ethics for public health," vol. 92, no. 7, p. 1057. [Online]. Available: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1447186/pdf/0921057.pdf

[15] S. Gao, Z. Han, C. Li, Q. Ye, and J. Jiao, "Real-time multipedestrian tracking in traffic scenes via an RGB-d-based layered graph model," vol. 16, no. 5, pp. 2814–2825.

[16] W. Choi, C. Pantofaru, and S. Savarese, "A general framework for tracking multiple people from a moving camera," vol. 35, no. 7, pp. 1577–1591.

[17] C. Vondrick, A. Khosla, H. Pirsiavash, T. Malisiewicz, and A. Torralba, "Visualizing object detection features," vol. 119, no. 2, pp. 145–158.

[18] P. Dollar, R. Appel, S. Belongie, and P. Perona, “Fast feature pyramids for object detection,” vol. 36, no. 8, pp. 1532–1545.

[19] S. A. Papert and M. Minsky, “The summer vision project,” p. 6. [Online]. Available: http://hdl.handle.net/1721.1/6125

[20] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L. E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney, “Stanley: The robot that won the DARPA grand challenge,” vol. 36, pp. 1–43.

[21] M. Bansal, S. H. Jung, B. Matei, J. Eledath, and H. Sawhney, “A real-time pedestrian detection system based on structure and appearance classification,” in *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 903–909.

[22] S. Deibel, *Executive Summary: The Python Software Foundation*. [Online]. Available: https://www.python.org/psf/summary/

[23] J. Fleetwood, “Public health, ethics, and autonomous vehicles,” vol. 107, no. 4, pp. 532–537.

[24] a. Ess, B. Leibe, K. Schindler, and L. Van Gool, “A mobile vision system for robust multi-person tracking,” pp. 1–8. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4587581

[25] M. Telles, *Python Power!: The Comprehensive Guide*. [Online]. Available: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:No+Title#0%5Cnhttp://books.google.com/books?hl=en&lr=&id=wbELAAAAQBAJ&oi=fnd&pg=PP8&dq=Python+Power!:+The+Comprehensive+Guide&ots=wz00ZeQ4IW&sig=GeXLzgvwDbvnjLDUcMxZrd3apaY

[26] S. Mallick, *Histogram of Oriented Gradients in OpenCV*. [Online]. Available: https://www.learnopencv.com/histogram-of-oriented-gradients/

[27] J. B. Garnett, T. M. Le, Y. Meyer, and L. A. Vese, “Image decompositions using bounded variation and generalized homogeneous besov spaces,” vol. 23, no. 1, pp. 25–56.

[28] William A. Hoff, *Introduction to Wavelets in Image Processing*. Colorado School of Mines, Department of Electrical Engineering and Computer Science. [Online]. Available: http://inside.mines.edu/whoff/courses/EENG510/lectures/24-Wavelets.pdf

[29] V. V. Dixit, S. Chand, and D. J. Nair, “Autonomous vehicles: Disengagements, accidents and reaction times,” vol. 11, no. 12.

[30] J. Robert Oppenheimer, *Address to the Association of Los Alamos Scientists*. [Online]. Available: http://www.atomicarchive.com/Docs/ManhattanProject/OppyFarewell.shtml

[31] I. R. A. S. J.-F. B. Edmond Awad, Sohan Dsouza, *Moral Machines*. [Online]. Available: http://moralmachine.mit.edu/

[32] P. Viola and M. Jones, “Robust real-time object detection,” vol. 57, no. 2, pp. 137–154. [Online]. Available: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Robust+Real-time+Object+Detection#0

[33] N. Drakos and R. Moore, *Python Library Reference*. [Online]. Available: https://web.archive.org/web/20070329061639/http://www.python.org/doc/2.5/lib/node951.html

[34] Cweiske, *Gradient2*. Wikipedia. [Online]. Available: https://en.wikipedia.org/w/index.php?title=File:Gradient2.svg

[35] M. L. Scott, *Programming Language Pragmatics*, 4th ed. Morgan Kaufmann.

[36] B. Venners, *The Making of Python*. [Online]. Available: https://www.artima.com/intv/pythonP.html

[37] S. Singh, “Critical reasons for crashes investigated in the national motor vehicle crash causation survey,” pp. 1–2.

[38] Y. Amit and P. Felzenszwalb, “Object detection,” in *Computer Vision*. Springer US, pp. 537–542. [Online]. Available: http://link.springer.com/10.1007/978-0-387-31439-6_660

[39] J. An, B. Choi, K. B. Sim, and E. Kim, “Novel intersection type recognition for autonomous vehicles using a multi-layer laser scanner,” vol. 16, no. 7.

[40] G. van Rossum, *A Brief Timeline of Python*. [Online]. Available: http://python-history.blogspot.com/2009/01/brief-timeline-of-python.html

[41] A. Rosebrock, *Image Pyramids with Python and OpenCV*. [Online]. Available: https://www.pyimagesearch.com/2015/03/16/image-pyramids-with-python-and-opencv/

[42] S. S. Schweber, *In the Shadow of the Bomb : Oppenheimer, Bethe, and the Moral Responsibility of the Scientist*, e-book ed. Princeton University Press.

[43] J. Zemek, V. Bílik, and L. Zákutná, “Effect of some aldoses on growth of saccharomyces cerevisiae inhibited with molybdenum,” vol. 20, no. 6, pp. 467–469.