Computer Vision

MInDS @ Mines

> Computer vision is the utilization of computers to extract or use information available in images or image data. Computer vision consists of several methods of interpreting image data; from loading images and converting them into their pixel information to more complex methods. Machine learning can be utilized in computer vision applications but is not required. In this lecture we will cover some classical approaches of computer vision. These approaches can extract information that is used as the input features for machine learning.

Computer Vision (CV) has a variety of applications such as optical character recognition, object identification, facial recognition, or robot navigation. Developing these applications in computer vision can utilize machine learning, however it does not have to. In this lecture, we will refer to all computer vision methods that do not require machine learning as "classical computer vision". Classical computer vision methods investigate the inherent pattern within images through the geometric structures that may exist. For example, when a high contrast occurs between two pixels that might be the beginning of one object and the end of another. There are a plethora of feature extraction methods that do not use machine learning, we will cover a subset of these in today's lecture. Before we move on to feature extraction methods, let's start by looking at how computers store image data.

Images

Images come in various resolutions and file type storage options but all images boil down to one key thing; identifying the color of each pixel in the image. We can represent the color of a pixel using a variety of methods. The most commonly used one is RGB which is red, green, and blue values. Each pixel gets three values, one for each of red, green, and blue, between 0 and 255 representing the amount of each in the pixel's color. This gives us the ability to represent over 16 million colors. Other color definition approaches include HSV which represents hue, saturation, and value, and YUV which represents luminance and difference in color from blue and red. HSV is commonly used in printing and YUV is used in TV data transmission. Both HSV and YUV were invented to reduce the amount of data required to represent the same colors. Despite them still requiring three values, the number of bits needed to represent the color can decrease. We may lose precision in the color values but the general image will still hold.

We use a number between 0 and 255 since we are classically using 8 bits to represent the value. More recently we have started using a larger number depending on the available bits.
Using RGB, we can represent $255 \times 255 \times 255 = 16,581,375$ colors.

Another approach that loses a significant amount of information from an image is grayscale. Using grayscale to represent an image, we lose a lot of the information in the color. We can now represent each pixel as a number between 0 and 255, where 0 is black and 255 is white.

Classical Computer Vision Features

In machine learning we may use the direct RGB values of an image, however with classical methods, there are many types of features that we can extract. First, there are two main approaches; tracking the changes in the pixel values across the image, and detecting edges/corners in the image. For tracking changes, we have two common methods; histogram of oriented gradients (HOG), and local binary patterns (LBP). For edge/corner detection we have several common methods; Harris corner detection, Shi-Tomasi Corner Detection, scale invariant feature transforms (SIFT), speeded up robust features (SURF), features from accelerated segment test (FAST), binary robust independent elementary features (BRIEF), and oriented FAST and rotated BRIEF (ORB).

Histogram of Oriented Gradients (HOG)

The histogram of oriented gradients (HOG) is a histogram with bins for the gradient direction (angle) with the bin height calculated as a sum of the gradient magnitudes. We can choose any number of bins for the angles and any function of the gradient magnitudes for bin height.

In practice, using 9 bins and an unsigned gradient combined with the sum of the gradient magnitudes produces good results, especially in human detection in images. We can calculate the image gradient in the x and y directions using,

$$\frac{\partial \mathsf{I}}{\partial x} = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \times \mathsf{I}, \tag{1}$$

$$\frac{\partial \mathsf{I}}{\partial y} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \times \mathsf{I}, \tag{2}$$

where $\times$ here represents applying a convolution to the image matrix of intensity values[1]. The result of applying these convolutions is a matrix of the same size.

The gradient's magnitude and direction can be calculated as,

$$G = \sqrt{(\frac{\partial \mathsf{I}}{\partial x})^2 + (\frac{\partial \mathsf{I}}{\partial y})^2}, \theta = \tan^{-1} \frac{\frac{\partial \mathsf{I}}{\partial y}}{\frac{\partial \mathsf{I}}{\partial x}}. \tag{3}$$

To use the HOG as a feature, we break up an image into smaller blocks of pixels and each block can then be represented by the 9 bin values. We can also then normalize all the blocks using a norm of the resulting image vector. The full image is represented using a smaller number of dimensions that represent the gradients and their direction in the image.

Local Binary Patterns (LBP)

Local binary patterns compare each point to its neighborhood and determine a new value based on that. For each point in the neighborhood, the 8 sur-

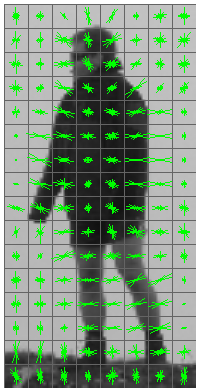

Figure 1: An example of HOG coupled with SVM applied to human detection..

[1] We can use the Y in YUV or the grayscale value to represent each pixel's intensity.

An example of calculating the x gradient given

$$\mathsf{I} = \begin{matrix} & \begin{matrix} 0 & 0 & 0 & 0 \end{matrix} & \\ \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \end{matrix} & \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix} & \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \end{matrix} \\ & \begin{matrix} 0 & 0 & 0 & 0 \end{matrix} & \end{matrix}$$

$$\frac{\partial \mathsf{I}}{\partial x} = \begin{bmatrix} 2 & 2 & 2 & 3 \\ 6 & 2 & 2 & 7 \\ 10 & 2 & 2 & 11 \\ 14 & 2 & 2 & 15 \end{bmatrix}$$

rounding points, we compare its value to its neighbors. Neighbors whose values are greater than the point are replaced by 0 and others are replaced by 1. We then rotate through the neighbors and record an 8 bit binary number that we then convert to decimal. This number is the new value for the pixel. If we break up the image into different blocks, we can represent each block as a 256[2] - dimension feature vector where each feature is the frequency that that number appears in the block.

An example of determining the value of a pixel given its neighborhood,

$$\mathsf{I} = \begin{bmatrix} 10 & 27 & 15 \\ 17 & 23 & 30 \\ 24 & 20 & 19 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 1 \\ 1 & & 0 \\ 0 & 1 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} & 165 & \end{bmatrix}$$

$$(10100101)_2 = 165$$

[2] 256 is the number of possible values to generate for each pixel since we have 8 neighbors and therefore 8 bits to represent the number.

Harris & Shi-Tomasi Corner Detection

Corners and edges in an image appear with a change in color intensity. To measure this intensity change, Harris corner detectors determine the change produced by a shift,

$$E(u,v) = \sum_{x,y} w(x,y) \times (I[x+u, y+v] - I[x,y])^2, \tag{4}$$

where u is the shift in the x direction, v is the shift in the y direction, $w(x,y)$ is a window function which weights the pixel values, and $I[a,b]$ is a lookup of the image intensity value at $x=a, y=b$. To maximize that function we get,

$$E(u,v) \sim \begin{bmatrix} u & v \end{bmatrix} \mathsf{M} \begin{bmatrix} u \\ v \end{bmatrix}, \mathsf{M} = \sum_{x,y} w(x,y) \begin{bmatrix} \mathsf{I}_x\mathsf{I}_x & \mathsf{I}_x\mathsf{I}_y \\ \mathsf{I}_y\mathsf{I}_x & \mathsf{I}_y\mathsf{I}_y \end{bmatrix}, \tag{5}$$

where $\mathsf{I}_x, \mathsf{I}_y$ are the gradients of the image matrix with respect to x and y. From this calculation we determine a score value,

$$R = \det(\mathsf{M}) - k\,\mathrm{tr}(\mathsf{M})^2 \tag{6}$$

$$= \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2, \tag{7}$$

where λ_1, λ_2 are the eigenvalues of M. We note that a small magnitude for the score results in a flat area, a large score results in a corner and a value close to 0 results in an edge.

Shi-Tomasi detectors build off of Harris corner detections in the calculations up to the score. The Shi-Tomasi method simply modifies the scoring function to be,

$$R = \min(\lambda_1, \lambda_2). \tag{8}$$

The same applies here with a low score resulting in a flat area, and a large score resulting in a corner. We can not directly use this score to determine the existence of edges.

Scale Invariant Feature Transforms (SIFT)

If we have two images of the same object at varying sizes, the previously discussed corner detection methods might not detect them as similar objects. SIFT[3] aims to deal with this issue. We will not cover SIFT in detail but give a quick overview of how it works. SIFT applies the following procedure:

[3]

1. Determine a scaled distribution of all points in the image. SIFT uses the difference of gaussian to find the scale-space.
2. Determine extreme points at each distribution within the image. Each sample point is compared to its eight neighbors in the current image and the 9 neighbors in a scale above and below.
3. Determine an orientation assignment for each point. This is done using a histogram of oriented gradients of the smoothed image, with 36 bins for 360° and using signed gradients.
4. Determine a feature vector for each key point. Each keypoint is then represented as a feature vector with 128 dimensions.

Speeded Up Robust Features (SURF)

SIFT features can be computationally costly to calculate. SURF[4] was introduced as a faster calculation for features with similar robustness to scale and orientation. Instead of using the difference of Gaussian, SURF uses the Box Filter method for approximation which can be calculated using integral images and can be done in parallel. SURF also uses a faster method for determining orientation. SURF are lower dimension features (64) compared to SIFT (128) and offer similar performance.

[4]

Features from Accelerated Segment Test (FAST)

Features from Accelerated Segment Test (FAST)[5] are created using machine learning. First, we compare each pixel to its 16 surrounding pixels' intensity based on a particular intensity threshold, t. We then determine if this point is a corner based on whether there are n contiguous points from its surroundings that are either brighter or darker than the pixel.

Afterwards, we use a decision tree algorithm to learn the representation based on entropy gain through categorizing the pixel as a corner or not.

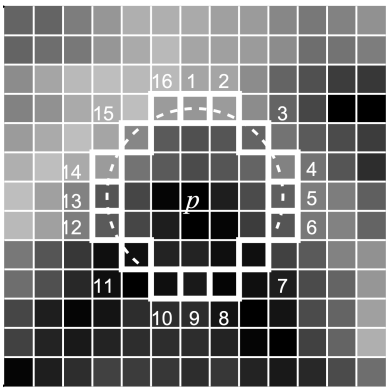

Figure 2: An example of FAST corner detection's neighborhood.

[5]

Binary Robust Independent Elementary Features (BRIEF)

Given a patch on a smoothed image, Binary Robust Independent Elementary Features (BRIEF)[6] are calculated by creating a binary number from whether or not the intensity of a pixel is higher than its surroundings. This binary number becomes the feature vector for any point. BRIEF does not create a feature vector independently but instead it works on top of other methods.

[6]

Oriented FAST and Rotated BRIEF (ORB)

ORB[7] is a method that was created as an open source alternative to both SIFT and SURF[8]. ORB simply combines the benefits from FAST and BRIEF while attempting to solve their drawbacks. The output method performs similar to SIFT and SURF.

[7]

[8] SIFT and SURF are both patented methods. They are very popular in academia and research but for any commercial application the authors must be compensated.