**Computer Vision: Advanced Image Processing all You Need.**

[[Koushik](https://medium.com/@koushikkushal95?source=post_page-----e903535440a5--------------------------------)](https://medium.com/@koushikkushal95?source=post_page-----e903535440a5--------------------------------)

[Koushik](https://medium.com/@koushikkushal95?source=post_page-----e903535440a5--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2F1670d68f972f&operation=register&redirect=https%3A%2F%2Fmedium.com%2F%40koushikkushal95%2Fbeyond-pixels-mastering-advanced-image-processing-in-computer-vision-e903535440a5&user=Koushik&userId=1670d68f972f&source=post_page-1670d68f972f----e903535440a5---------------------post_header-----------)

10 min read

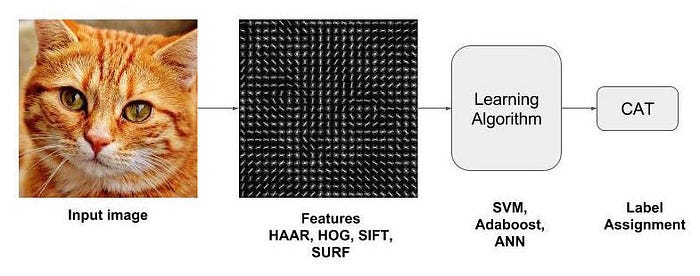
·

Dec 10, 2023

125

1

Unraveling the Mathematics of Image Processing for Enhanced Computer Vision



Feature Extraction. source: [Satya Mallick](https://learnopencv.com/image-recognition-and-object-detection-part1/)

In computer vision, feature extraction plays a pivotal role in transforming raw input data, such as images, into a format that is more amenable to analysis and understanding. Feature extraction involves identifying and selecting relevant information or features from the input data. These features capture distinctive patterns, structures, or characteristics of the data, making it easier for algorithms to interpret and make decisions. The primary purposes of feature extraction in computer vision include:

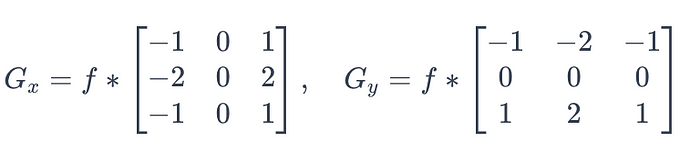
1. **Dimensionality Reduction:** Feature extraction often reduces the dimensionality of the data by representing it in a more compact and meaningful form. This reduction facilitates more efficient processing and storage of information.
2. **Highlighting Relevant Information:** Features extracted from images highlight specific aspects of interest, such as edges, corners, textures, or key points. These features are critical for tasks like object recognition, image classification, and scene understanding.
3. **Enhancing Robustness:** By focusing on essential features, computer vision algorithms become more robust to variations in lighting conditions, viewpoint changes, and other factors that can affect the appearance of objects in images.
4. **Enabling Discrimination:** Extracted features serve as discriminative elements that distinguish between different objects or classes. They capture unique aspects of the data that contribute to accurate identification and classification.
5. **Preparing Data for Machine Learning:** Feature extraction is a crucial step in preparing data for machine learning models. By representing images with relevant features, the learning algorithms can better generalize patterns and relationships within the data.
6. **Improving Computational Efficiency:** Extracting relevant features reduces the computational load on subsequent processing steps. It allows algorithms to focus on the most informative aspects of the data, leading to faster and more efficient computations.

Let’s delve deeper into the theoretical aspects of some feature extraction methods:

**Edge Detection:**

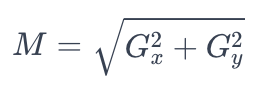
Edge detection aims to identify regions in an image where there is a significant change in intensity. Edges often represent object boundaries or important structures within the image. The Canny edge detector is a popular method that involves multiple steps, including gradient calculation, non-maximum suppression, and edge tracking by hysteresis.

Edge detection often involves computing the gradient of the image intensity. The gradient (∇*f*) of an image *f*(*x*,*y*) is computed using convolution with derivative filters (e.g., Sobel or Prewitt operators):



Gradient derivative filters. source: author

The magnitude of the gradient (*M*) is calculated as:



Calculate Gradient Magnitude

and the direction (*θ*) is determined as:



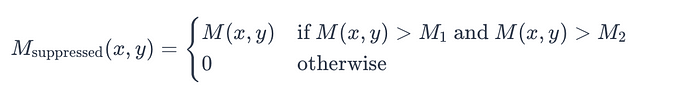
The Canny edge detector further involves non-maximum suppression and hysteresis thresholding. Let’s deep dive how it functions.

**Non-Maximum Suppression:**

After computing the gradient magnitude (*M*) and direction (*θ*) for each pixel, non-maximum suppression is applied to thin the edges and keep only the local maxima in the gradient direction. The idea is to ensure that the detected edges have a clear and sharp response along the gradient direction.

For each pixel, non-maximum suppression involves comparing the gradient magnitude of the pixel with its neighbors in the gradient direction. If the gradient magnitude at the pixel is greater than its neighbors, it is retained; otherwise, it is suppressed.

Let *M*(*x*,*y*) be the gradient magnitude, and *θ*(*x*,*y*) be the gradient direction at pixel (*x*,*y*). The non-maximum suppression can be expressed as:



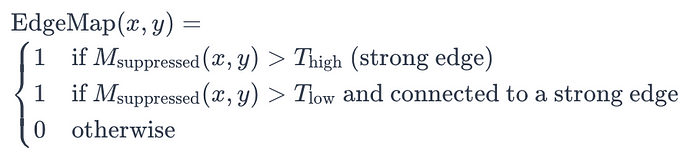
Here, *M*1​ and *M*2​ are the gradient magnitudes of the two neighboring pixels along the gradient direction.

**Hysteresis Thresholding:**

Hysteresis thresholding is applied to distinguish between strong edges, weak edges, and noise. It involves setting two threshold values: a high threshold (*T*high​) and a low threshold (*T*low​). Pixels with gradient magnitudes above *T*high​ are considered strong edges, while pixels between *T*low​ and *T*high​ are considered weak edges.

The algorithm then traces along the strong edges and connects weak edges to the strong edges if they form a continuous path. This helps in preserving edges while suppressing noise.

hysteresis thresholding can be expressed as:



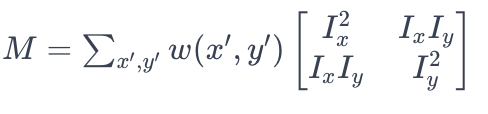
This process is typically implemented using depth-first search or similar techniques to trace and connect the weak edges forming a coherent edge map.

In summary, non-maximum suppression ensures that only local maxima in the gradient direction are retained, and hysteresis thresholding helps in distinguishing strong edges from weak edges and connecting them to form a more robust edge map.

**Corner Detection (Harris Corner Detection):**

**Structure Tensor Calculation:**

For a given pixel (*x*,*y*), the first step is to calculate the structure tensor *M*, which is a 2x2 matrix representing local intensity changes in the image. The structure tensor is defined as:

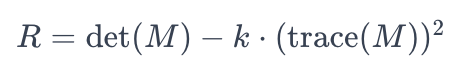


Tensor Calculation

where *Ix*​ and *Iy*​ are the image gradients at pixel (*x*,*y*), and *w*(*x*′,*y*′) is a window function centered at (*x*,*y*).

**Corner Response Function:**

The corner response function *R* is computed from the eigenvalues (*λ*1​ and *λ*2​) of the structure tensor *M*:

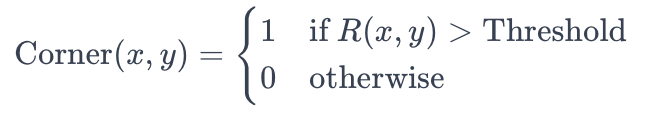


Corner response function

Here, det(*M*) is the determinant of *M*, trace(*M*) is the trace of *M*, and *k* is an empirically determined constant (typically in the range of 0.04 to 0.06).

**Corner Identification:**

After computing the corner response function for each pixel, corners are identified by selecting pixels with high corner response values. A common approach is to threshold the corner response function and consider pixels with response values above a certain threshold as corners



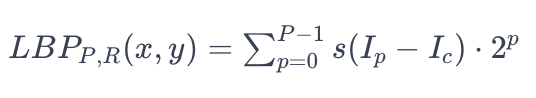
Corner identification

**Texture Analysis**

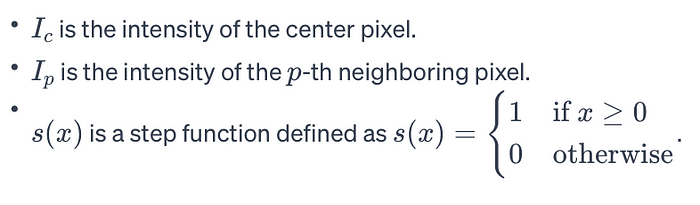
Local Binary Patterns (LBP) is a texture analysis method that captures the local patterns of pixel intensities in an image. It is particularly useful for characterizing textures and patterns in images. Here’s a detailed explanation of LBP:

For a given pixel (*x*,*y*), LBP is calculated by comparing the intensity of the center pixel with the intensities of its neighboring pixels. The binary pattern is generated by assigning a value of 1 to a neighboring pixel if its intensity is greater than or equal to that of the center pixel; otherwise, a value of 0 is assigned. This process is repeated for all neighbors in a predefined circular neighborhood.

Let *P* be the number of sampling points in the neighborhood, and *R* be the radius of the circular neighborhood. The LBP value for a pixel (*x*,*y*) is computed as follows:



Here,



**Color Histogram**

A color histogram represents the distribution of color intensities in an image. It provides a quantitative description of the colors present in the image, which can be useful for various computer vision tasks.

**Color Space Conversion:**

Before computing the color histogram, the image is typically converted to a color space that separates its color channels. Common color spaces include RGB (Red, Green, Blue), HSV (Hue, Saturation, Value), or LAB (CIELAB). Let’s assume RGB for this explanation.

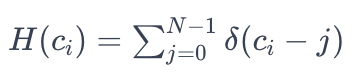
**Binning:**

In a color histogram, the color space is divided into a set of bins or discrete intervals. Each bin corresponds to a range of color intensities. The number of bins determines the granularity of the histogram. For example, if we use 256 bins for each color channel in the RGB space, we cover all possible intensities (0 to 255).

**Calculation of Histogram:**

For each pixel in the image, the color values are quantized into the corresponding bins. The count of pixels falling into each bin is accumulated to form the histogram.

Mathematically, let *H*(*ci*​) be the histogram for color channel *i*, where *i* can be red (R), green (G), or blue (B). The histogram is calculated as:



Color quantisation of each channel

Here:

* *N* is the number of bins.
* *δ* is the Dirac delta function, which equals 1 when *ci*​ falls within the *j*-th bin, and 0 otherwise.

**Histogram of Oriented Gradients (HOG):**

Histogram of Oriented Gradients (HOG) is a feature descriptor widely used in computer vision for object detection. It captures information about the local gradient directions in an image. The HOG algorithm involves the following steps:

For each pixel in the image, compute the **gradient magnitude** and orientation. This can be done using convolution with Sobel filters or any other gradient calculation method. Follow the earlier explanation of **edge detection**and compute G: gradient magnitude, and *θ:*gradient orientation.

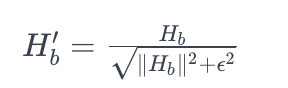
Divide the image into small cells (e.g., 8x8 pixels). Each cell contains a local histogram of gradient orientations. For each cell, calculate a histogram of gradient orientations. The histogram is created by accumulating the gradient magnitudes into bins based on their orientations. Group cells into larger blocks (e.g., 2x2 cells). Normalize the histograms within each block to improve invariance to changes in illumination and contrast. Concatenate the normalized histograms from all blocks to form the final HOG descriptor for the image.

**Mathematically:**

Let *M* be the number of cells, *N* be the number of bins in the histogram, and *B* be the number of blocks. The HOG descriptor *H* is formed as follows:

*H*=[*H*1​,*H*2​,…,*HB*​]

where each *Hb*​ is the normalized histogram for block *b*. The normalization is often done using the L2-norm:



Here, *ϵ* is a small constant added to the denominator to avoid division by zero.

The HOG descriptor can be used for various computer vision tasks, such as object detection and pedestrian recognition.

**Principal Component Analysis (PCA)**

PCA is a dimensionality reduction technique used to transform high-dimensional data into a lower-dimensional space while retaining the most important information. It is commonly applied to image data, feature vectors, or any dataset where reducing dimensionality is beneficial.

PCA begins by mean-centering the data, ensuring that the mean of each feature is zero. Calculate the covariance matrix to understand the relationships between different features. Find the eigenvectors and eigenvalues of the covariance matrix. Select the top *K* eigenvectors corresponding to the largest eigenvalues. Sort the eigenvalues in descending order, and choose the top *K* eigenvectors to form the matrix. Project the mean-centered data onto the subspace spanned by the selected principal components.

**Key Concepts:**

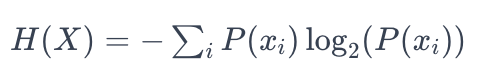
* Eigenvalues and Eigenvectors: Eigenvalues represent the amount of variance along the corresponding eigenvector direction. Larger eigenvalues indicate directions with more significant variability.
* Principal Components: Principal components are the eigenvectors of the covariance matrix. Each principal component represents a direction in the original feature space.
* Variance Retention: PCA allows the user to choose the number of principal components to retain, based on the desired amount of variance to be preserved in the data.
* Dimensionality Reduction: The dimensionality of the data is reduced from *D* to *K* dimensions (*K*≤*D*).

PCA is a valuable tool for preprocessing data, reducing noise, and improving the efficiency of machine learning algorithms, especially when dealing with high-dimensional datasets.

**Entropy**

Shannon Entropy, named after Claude Shannon, is a measure of uncertainty or information content in a set of data. In the context of information theory, it quantifies the average amount of surprise or unpredictability associated with the outcomes of a random variable.

The formula for Shannon Entropy (*H*) for a discrete random variable with probability distribution *P*(*x*) is given by:



Shannon’s Entropy

Here:

* *xi*​ represents each possible outcome of the random variable *X*.
* *P*(*xi*​) is the probability of occurrence of *xi*​.
* The sum is taken over all possible outcomes.

**Interpretation:** Higher entropy indicates higher uncertainty or disorder in the data, while lower entropy suggests more predictability.

**Units:** Entropy is measured in bits if the logarithm is base 2. In practice, the base of the logarithm depends on the unit of measurement for the probabilities (e.g., natural logarithm for base e*e* or base 10 logarithm).

**Entropy in Information Theory:** In information theory, entropy is used to quantify the average number of bits needed to represent the information content of a message or the average “surprise” associated with receiving a symbol from a source.

In the context of images, Shannon Entropy can be used to measure the amount of information or complexity in the pixel intensities. Images with more uniform pixel intensities across the image may have lower entropy, while images with varied intensities may have higher entropy.

Apply feature extraction methods as a preprocessing step before training machine learning models. Ensure that the selected features align with the requirements of the specific task. Integrate the extracted features as input to machine learning models (e.g., SVM, Random Forest, Neural Networks). Fine-tune hyperparameters based on the characteristics of the extracted features.

In summary, feature extraction is a crucial step in the machine learning pipeline, enhancing the model’s ability to generalize and perform well on diverse datasets. The choice of feature extraction method depends on the characteristics of the data and the specific requirements of the machine learning task.

The code will be available this [github](https://github.com/Koushikl0l/Writings_/tree/main) repository soon.

Thank You!