Fondamental of Data Science, 1st semester at Sapienza, Pt1

Emanuele Iaccarino

October 2024

Contents

Chapter 1

HomeWork1: Image Filtering and Object Identification

1.0.1 Historical Background

A foundational model in computer vision is the **pinhole camera model**, which represents how images are formed in a camera. It can be conceptualized as a box with a small hole that lets light in, projecting an inverted image on the opposite side. This principle has been used by both artists and scientists for centuries.

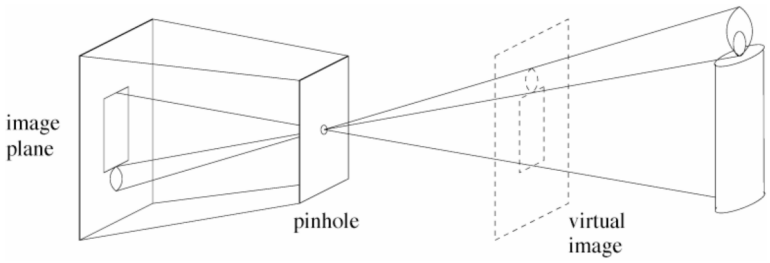


Figure 1.1: Pinhole Camera Model Illustration

In this setup, light enters a darkened room through a small hole, forming a clear but inverted image of the outside world. This is important for understanding how early attempts at capturing and studying images evolved into modern computational techniques.

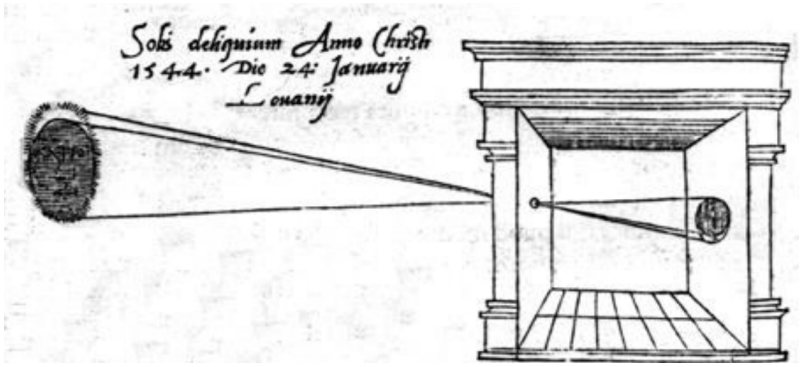


Figure 1.2: Image Formation Process

In modern digital imaging, the process involves converting the physical image captured by a camera into a digital image made up of pixels, each pixel representing a small portion of the image in terms of light intensity (grayscale) or color. Understanding how this digitization process works is fundamental to both image processing and computer vision.

1.0.2 Object Recognition

One of the major goals of computer vision is object recognition, which involves teaching computers to recognize objects from an array of pixel values. This problem is usually referred to as inverse graphics, where the computer has to work backward from pixel data to recognize shapes and objects.

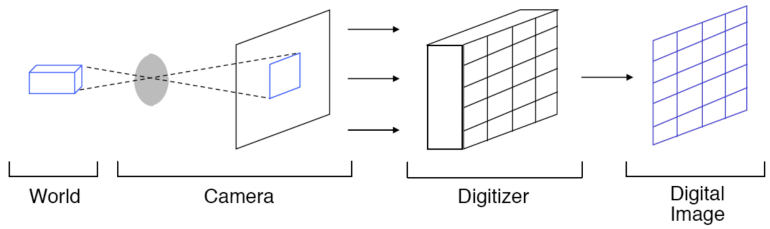


Figure 1.3: Object Recognition Pipeline

1.0.3 Depth Perception

Another key goal is depth perception, where a computer tries to infer the 3D structure of a scene from a flat, 2D image.

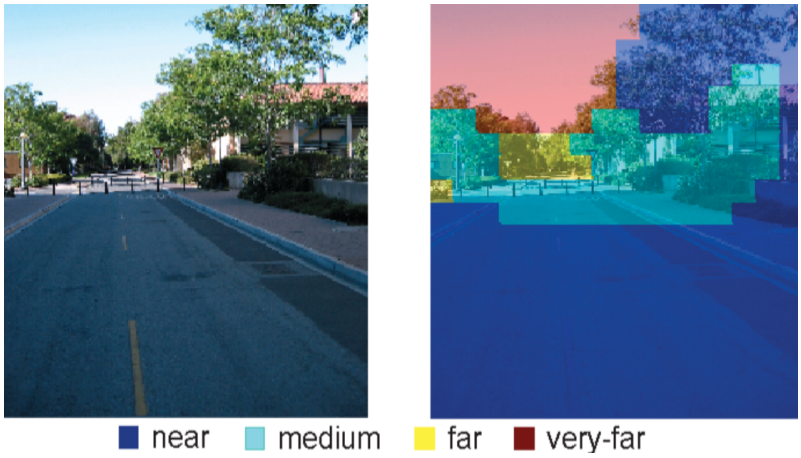


Figure 1.4: Depth Perception Using Visual Cues

A key case study that demonstrates this is 3D structure recovery, where cues like shading, color, and shadows are used to infer depth in an image. Artists have used these visual techniques in trompe l'oeil paintings to create optical illusions of depth on flat surfaces.

1.0.4 Color and Shading

Color and shading play critical roles in how humans and machines perceive depth and shapes. Changes in shadow or lighting can dramatically alter depth perception, and these concepts are applied in computational models for tasks like 3D reconstruction or object recognition.

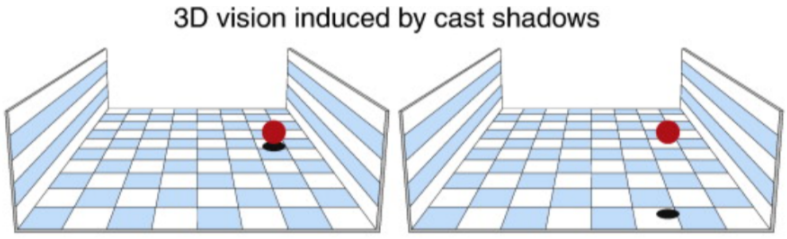


Figure 1.5: Role of Color and Shading in Depth Perception

In the realm of **computer vision**, object recognition becomes highly complex due to variations in lighting, ambiguities in the image, so objects might appear different under various conditions, leading to challenges in recognizing them accurately. A critical aspect of improving object recognition is the role of context and prior expectations: context significantly helps narrow down the possibilities of what an object could be, based on its surroundings. Similarly, prior knowledge or expectation helps in recognition tasks. For instance, in Giuseppe Arcimboldo's paintings, where human faces are formed from fruits, our expectations play a big role in perceiving the image as a face.

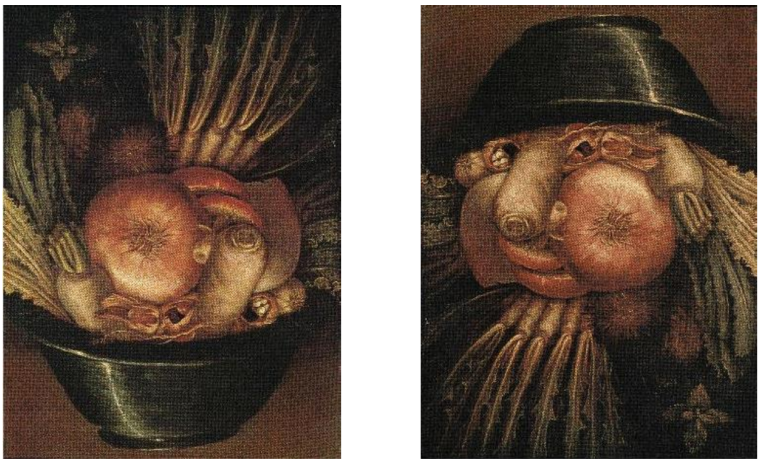


Figure 1.6: Giuseppe Arcimboldo's paintings

1.0.5 Recognition, Localization, and Segmentation

In computer vision, recognition, localization, and segmentation are key tasks:

Recognition Recognition involves identifying what an object is, using either object identification (recognizing specific objects like your own dog) or object classification (recognizing categories like any dog or apple).

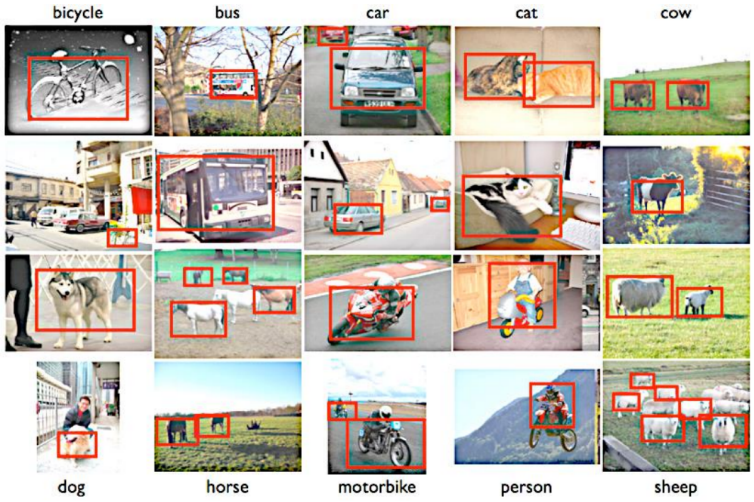


Figure 1.7: Recognition Example

Localization Localization estimates the position and orientation of the object in a scene.

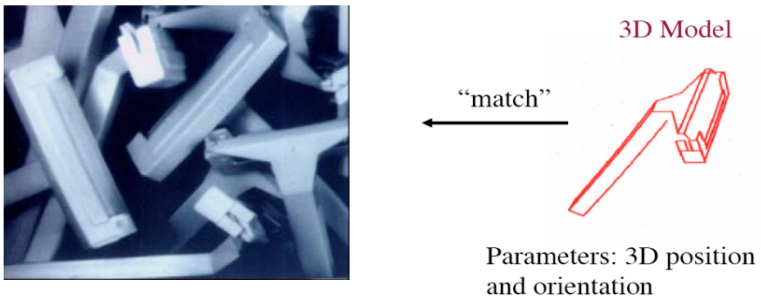


Figure 1.8: Localization Example

Segmentation Segmentation separates the object from its background, helping to isolate it for further analysis.

Figure 1.9: Segmentation Example

1.0.6 Challenges in Object Recognition

Recognition tasks can be complicated by several factors, such as:

- **Multi-scale:** Objects can appear in various sizes due to their distance from the camera.
- **Multi-view:** Objects can be viewed from different angles.
- **Occlusion:** Parts of the object may be hidden.
- **Clutter:** There could be noise or irrelevant objects in the background.
- **Illumination:** Changes in lighting can make it hard to identify objects.
- **Articulation:** Objects can change form, such as a person moving their arms.

To handle these challenges, digital image filtering is crucial.

1.1 Basics of Digital Image Filtering

Computer Vision: Involves "reversing" the imaging process to extract useful information from images. This includes 2D digital image processing and 3D image analysis for pattern recognition and understanding.

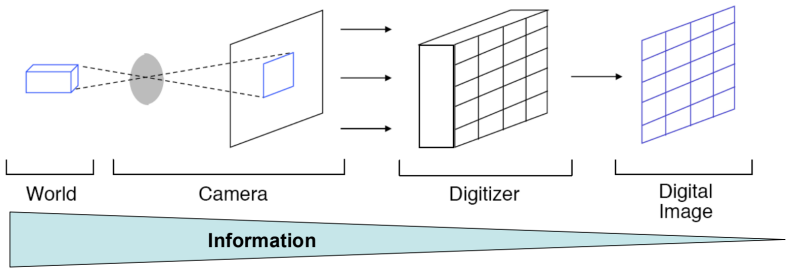


Figure 1.10: Computer Vision process

1.2 2D Convolution (Discrete)

Digital image filtering refers to applying a function to a local image patch (like a 3x3 block) to enhance the image or extract features.

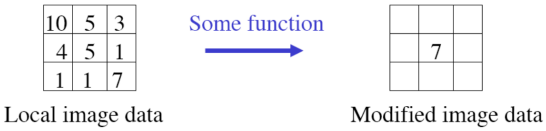


Figure 1.11: Example of Digital Image Filtering

1.2.1 2D Signals and Convolution

Convolution is a key operation in image filtering. It involves taking an image and applying a filter (kernel) to extract specific features like edges or reduce noise.

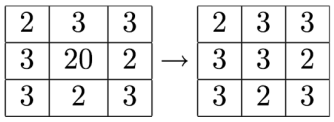


Figure 1.12: 2D Convolution Process

A convolution operation in 2D can be seen as a matrix multiplication process:

$$f[m,n] = (I \otimes g) = \sum_{k,l} I[m-k, n-l]g[k,l]$$

Where:

- $I[m,n]$ is the discrete image matrix.
- $g[k,l]$ is the filter or kernel.
- $f[m,n]$ is the resulting filtered image.

Illustration of 2D Convolution

$$I[k,l] = \begin{pmatrix} 8 & 5 & 2 \\ 7 & 5 & 3 \\ 9 & 4 & 1 \end{pmatrix} \quad \otimes \quad g[k,l] = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} \quad \Rightarrow \quad f[m,n] = 18$$

The process involves:

1. **Mirror the Filter:** The kernel $g[k,l]$ is flipped both horizontally and vertically to create a mirrored version.
2. **Swipe Across the Image:** The mirrored filter is moved across the image pixel by pixel.
3. **Multiply and Sum:** For each overlapping position, the filter values are multiplied with the corresponding image pixel values, and the results are summed.
4. **Resulting Pixel Value:** The final value of the filtered pixel at (m,n) is the sum of all these products.

$$\text{At position } (m,n): \sum_{k=-1}^{1} \sum_{l=-1}^{1} I[m-k, n-l]g[k,l] = f[m,n]$$

Example of multiplication and summation for a pixel:

$$f[m,n] = (-1\times9)+(0\times4)+(1\times1)+(-1\times7)+(0\times5)+(1\times3)+(-1\times8)+(0\times5)+(1\times2)$$

$$= (-9) + (0) + (1) + (-7) + (0) + (3) + (-8) + (0) + (2) = 18$$

1.3 Linear Filtering

Linear filtering is a basic operation in image processing where each pixel in an image is replaced by a linear combination of its neighboring pixels.

1.3.1 Linear Filter Examples

Filtered (no changes)

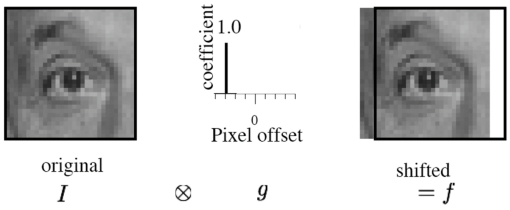


Figure 1.13: Filtered (no changes)

The coefficient plot in the middle illustrates the filter's effect, with a peak at pixel offset 0, meaning no modification was applied (filter is an identity filter).

Shifted

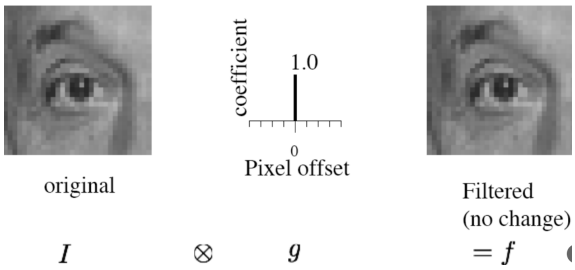


Figure 1.14: Shifted

Filter, with a non-zero offset cause a shift in the image.

Blurred

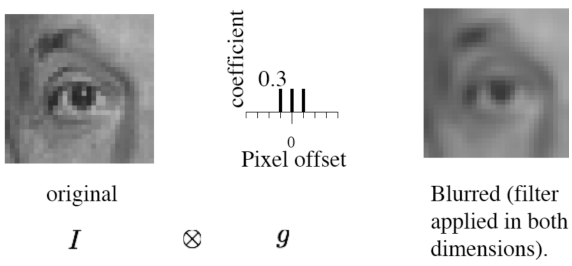


Figure 1.15: Blurred

1.4 Definition of an Image (Extra topic 04.10.24)

An image can be represented as a function that maps locations (coordinates) to pixels. To represents the image at location (x, y) we use:

$$F(x, y)$$

For an RGB image, the function can be written as:

$$f(x, y) = \begin{bmatrix} r(x, y) \\ g(x, y) \\ b(x, y) \end{bmatrix}$$

where each color channel (red, green, blue) is a value between 0 and 255.

- corresponds to black, while 255 corresponds to white.
- A single pixel can have up to 16,777,216 combinations ($256 \times 256 \times 256$), considering 8 bits for each color channel.

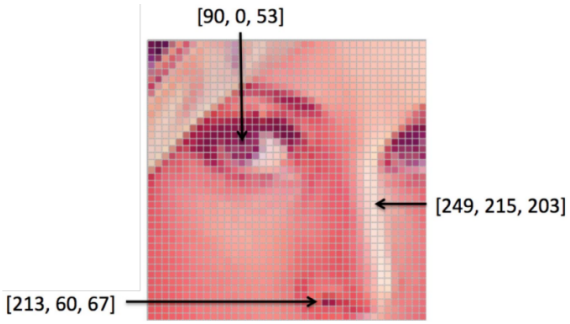


Figure 1.16: Colored img

- In the case of a grayscale image, each pixel has values ranging from 0 to 255, where the range represents the intensity of light, from black to white.

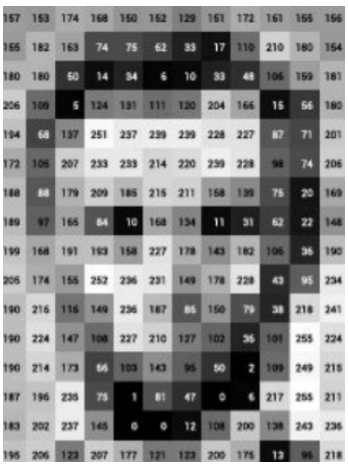


Figure 1.17: Grayscale img

Filter A filter is defined as a function applied to the pixels of an image, $h(f(x, y))$.

This function can modify the pixels for various purposes such as enhancement, smoothing, and noise reduction.

1.5 Convolution

Convolution is a mathematical operation applied to an image using a small matrix called a kernel or filter.

- The kernel is passed over the image, performing element-wise multiplication followed by summation over the region of interest.
- The result is stored in the corresponding pixel in the output image.

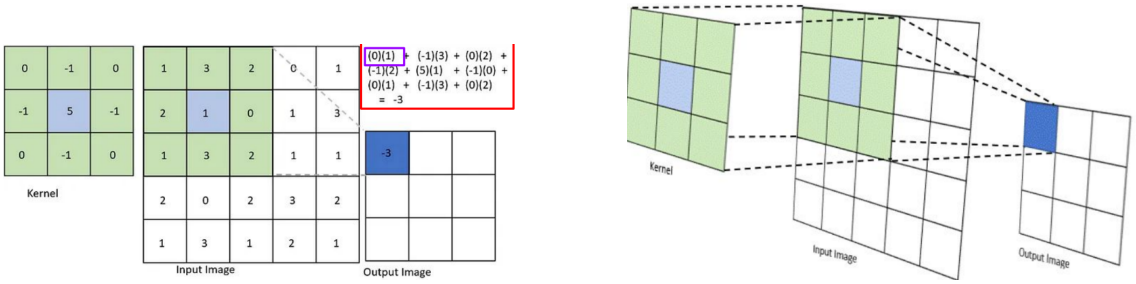


Figure 1.18: Practical example of convolution

1.5.1 2D Convolutions

The general formula for a 2D convolution is:

$$y(n, m) = \sum_k \sum_l x(k, l) \cdot k(n - k, m - l)$$

where $F(x, y)$ is the image and $h(m, n)$ is the kernel.

Padding

Padding is often added to the image before convolution:

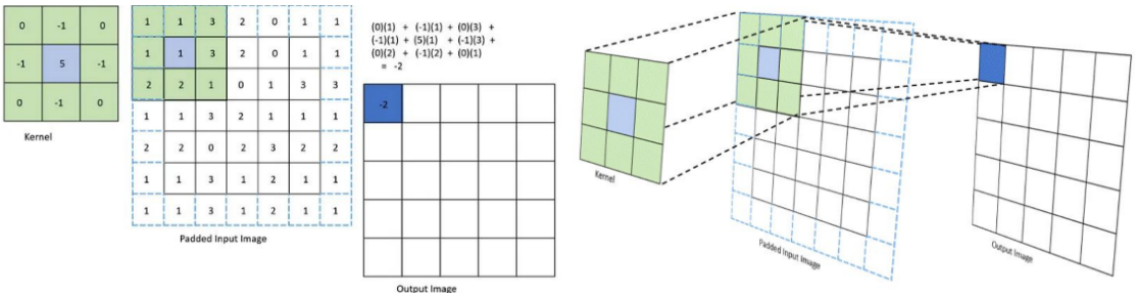


Figure 1.19: Padding practical example

- **Zero-padding**: Padding with zeros around the border of the image. It helps preserve the size of the output image compared to the input, particularly when using smaller kernels.
- If no padding is applied, the output image will shrink after each convolution operation, focusing less on the border pixels.

Key Properties of Convolutions

- **Feature Extraction**: Convolution helps detect local patterns in an image (like edges, textures). In deep learning, convolutional layers automatically learn these features at different levels of abstraction.
- **Localized Operations**: Convolutions operate on small, localized regions of the image (based on the size of the kernel), making it efficient for tasks like smoothing, sharpening, or edge detection.
- **Shift Invariance**: Convolutions preserve the spatial structure of the image, allowing the detection of features irrespective of their position. This is crucial for applications like object recognition.

1.6 Common Examples of Filters

Here are some common filters used in image processing:

1.6.1 Identity Filter

An identity filter is a 3x3 matrix with 1 at the center and 0 elsewhere. It leaves the image unchanged as it multiplies each pixel by 1 and adds 0 to its neighbors.

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

Figure 1.20: Identity Filter

1.6.2 Sharpen Filter

A sharpen filter enhances edges by amplifying the difference between a pixel and its neighbors. It uses a kernel with a larger central value and negative values around it.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Figure 1.21: Sharpen Filter Kernel

1.6.3 Mean Blur (Box Filter)

A mean blur averages the values of each pixel with its neighbors. The kernel used is typically filled with equal weights, resulting in a blurred image with reduced fine details.

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

Figure 1.22: Mean Blur (Box Filter)

1.6.4 Gaussian Blur

Similar to mean blur but uses a Gaussian distribution to weight the neighboring pixels. It gives more emphasis to nearby pixels and creates a more natural-looking blur.

$$\begin{bmatrix} 1/16 & 2/16 & 1/16 \\ 2/16 & 4/16 & 2/16 \\ 1/16 & 2/16 & 1/16 \end{bmatrix}$$

Figure 1.23: Mean Blur (Box Filter)

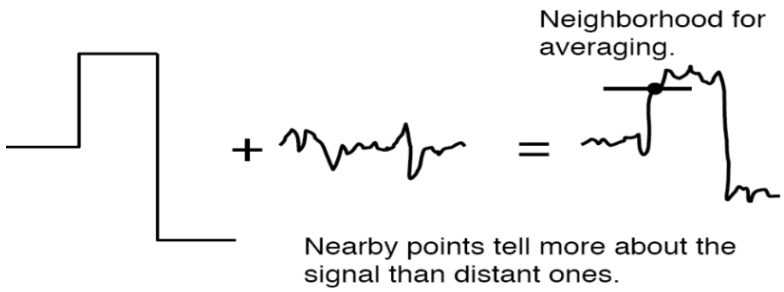
1.7 Signal and Noise in Images

In image processing, an image I can be modeled as a sum of signal S and noise N, i.e.,

$$I = S + N$$

Where:

- S is the true image signal.
- N is the noise, which is unwanted random variations that interfere with the signal



Noise is independent of the signal and does not provide useful information about the image content. In practice, noise is often modeled as having an expected value of zero, $E(N_i) = 0$, and being independent across pixels.

- The signal s_i is deterministic.
- For noise n_i and n_j, we have:

$$n_i \perp n_j \quad \text{for} \quad i \neq j$$

- Moreover, n_i and n_j are independent and identically distributed (i.i.d.), i.e.,

$$n_i, n_j \overset{\text{i.i.d.}}{\sim} N(0, \sigma^2)$$

1.7.1 Averaging Filter

An averaging filter replaces each pixel in an image with the average of its surrounding neighbors using a filter mask with positive entries that sum to 1.

Figure 1.24: Enter Caption

1.7.2 Gaussian Averaging (Isotropic Gaussian Filter)

The Gaussian filter is a smoothing filter that gives more weight to nearby pixels using a Gaussian function. This helps to reduce noise while preserving important features.

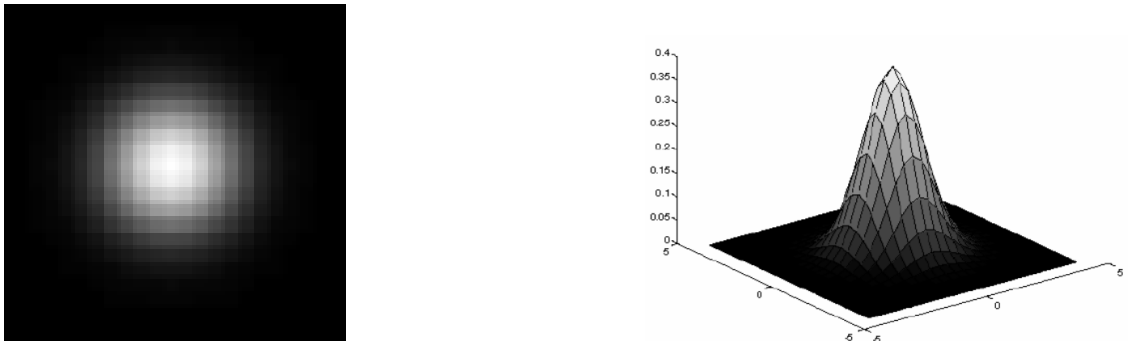


Figure 1.25: Gaussian filter example

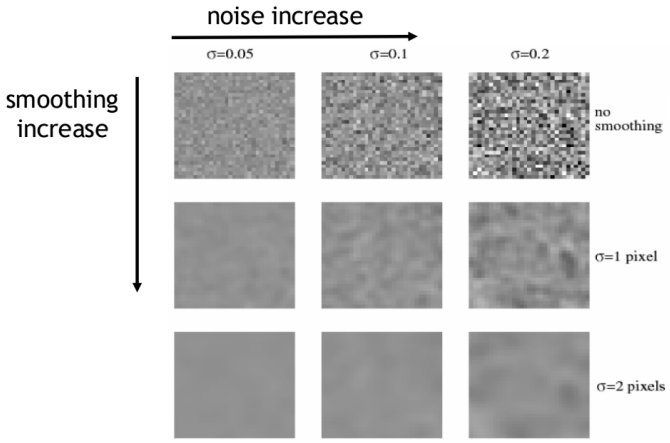
The degree of smoothing is controlled by the standard deviation σ of the Gaussian function.

$$G(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

1.7.3 Smoothing with a Gaussian Filter

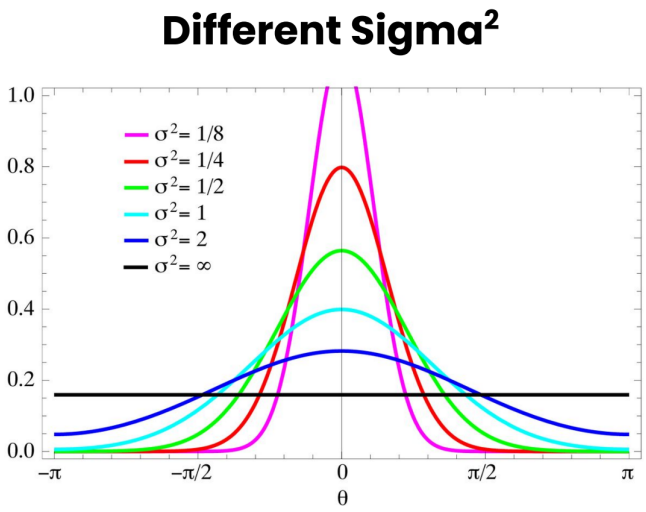
Noise and smoothing are controlled by the standard deviation σ of the Gaussian filter.

- Larger values of σ result in more smoothing and less detail in the image.
- Smaller σ preserves more original detail but reduces less noise.



The degree of smoothing is determined by the variance σ^2, where:

- Smaller σ^2 sharpens the image.
- Larger σ^2 blurs the image, smoothing over a wider area.



1.7.4 Efficient Implementation (Separability)

Separable Filters

Both the Box Filter and the Gaussian Filter can be separated into two 1D filters, which allows for more efficient computation. Instead of applying a 2D filter to the image directly, you can first convolve the image along the rows with a 1D filter, then convolve the result along the columns with the same 1D filter.

$$(f_x \otimes f_y) \otimes I = f_x \otimes (f_y \otimes I)$$

Example: Separable Box Filter

An example of a separable box filter is illustrated below. Instead of applying the 3x3 filter directly to the image, it can be decomposed into two 1D filters.

$$f_x \otimes f_y = \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix} = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix} \otimes \begin{bmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{bmatrix}$$

This separability allows us to perform convolution more efficiently by applying two 1D filters, one along the rows and the other along the columns.

1.7.5 Gaussian Filtering and Separability

Gaussian Kernel Formula

The 2D Gaussian kernel (so the Gaussian in both directions) is expressed as:

$$g(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

where:

- σ is the standard deviation,
- x and y are the horizontal and vertical distances from the center of the kernel.

Definition

A 2D Gaussian filter can be separated into two 1D Gaussian filters, which allows us to break the convolution process into two simpler 1D convolutions.

Mathematical Explanation step by step

The full 2D convolution of the image with a 2D Gaussian filter is expressed as:

$$h(i, j) = \sum_{k=1}^{m} \sum_{l=1}^{n} g(k, l) f(i - k, j - l)$$

where:

- $h(i, j)$ is the output image at position (i, j),

- $f(i-k, j-l)$ is the input image at position $(i-k, j-l)$,
- $g(k, l)$ is the kernel (or Gaussian filter) value at (k, l).

This means that the output pixel $h(i, j)$ is obtained by taking a weighted sum of its neighbors in the input image $f(i-k, j-l)$, where the weights are given by the Gaussian kernel $g(k, l)$.

Step 1: Expressing the 2D Gaussian Kernel

The 2D Gaussian kernel is rotationally symmetric and depends on both the horizontal offset k and the vertical offset l. It is given by the formula:

$$g(k, l) = e^{-\frac{k^2+l^2}{2\sigma^2}}$$

Here, σ is the std that controls the spread of the Gaussian function.

Step 2: Gaussian Separability

Since 2D Gaussian filter is separable, it can be factored into two 1D Gaussian filters: one that depends only on the horizontal direction k, and another that depends only on the vertical direction l.

Mathematically, this separability is expressed as:

$$g(k, l) = e^{-\frac{k^2}{2\sigma^2}} \cdot e^{-\frac{l^2}{2\sigma^2}}$$

Step 3: Horizontal Convolution

With separability, instead of performing a 2D convolution, we first apply a 1D convolution along the rows (horizontal direction). This step calculates an intermediate result $h'(i, j)$, which smooths the image horizontally:

$$h'(i, j) = \sum_{l=1}^{n} e^{-\frac{l^2}{2\sigma^2}} f(i, j-l)$$

This operation reduces the image only in the horizontal direction, using the 1D Gaussian filter applied to each row.

Step 4: Vertical Convolution

Once the horizontal convolution is done, we apply a second 1D convolution along the columns (vertical direction), using the intermediate image $h'(i, j)$ from the previous step. This completes the smoothing of the image in the vertical direction:

$$h(i, j) = \sum_{k=1}^{m} e^{-\frac{k^2}{2\sigma^2}} h'(i-k, j)$$

This step processes the intermediate image vertically, producing the final smoothed image $h(i, j)$.

Efficiency of Separability

By applying the convolution in two 1D steps—first along the rows and then along the columns—the total computational complexity is significantly reduced. Instead of performing $m \times n$ operations per pixel for a full 2D convolution, we now perform $m + n$ operations per pixel, making the algorithm much faster and more efficient.

1.8 Multi-Scale Image Representation

In object detection and recognition tasks, objects can appear at different sizes in an image. A multi-scale representation helps search for objects more effectively, allowing the algorithm to detect objects regardless of their size or resolution.

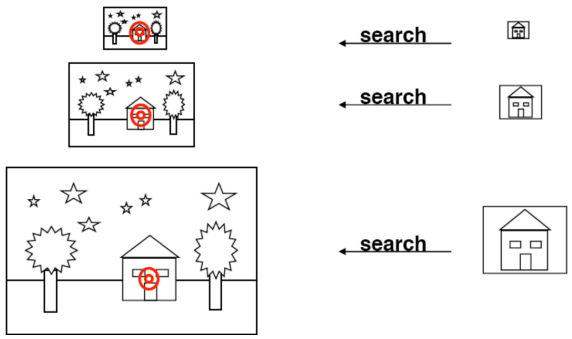


Figure 1.26: Object detection across scales.

Key Considerations

- **Information preserved across scales**: Larger, more generalized features of the image are retained as the scale decreases (e.g., general shapes and larger areas of contrast).
- **Information lost across scales**: Fine details and small features are lost as the image is blurred and downsampled.

1.8.1 Gaussian Pyramid

The **Gaussian Pyramid** is a technique used to represent images at multiple scales by progressively blurring the image and reducing its size. The process consists of two main operations:

- **Gaussian Blurring**: This operation smooths high-frequency details (e.g., sharp edges).
- **Downsampling**: This reduces the resolution by removing pixels from the image.

By repeating these steps, the image becomes smaller and smoother at each level of the pyramid, which is useful for detecting features at different scales.

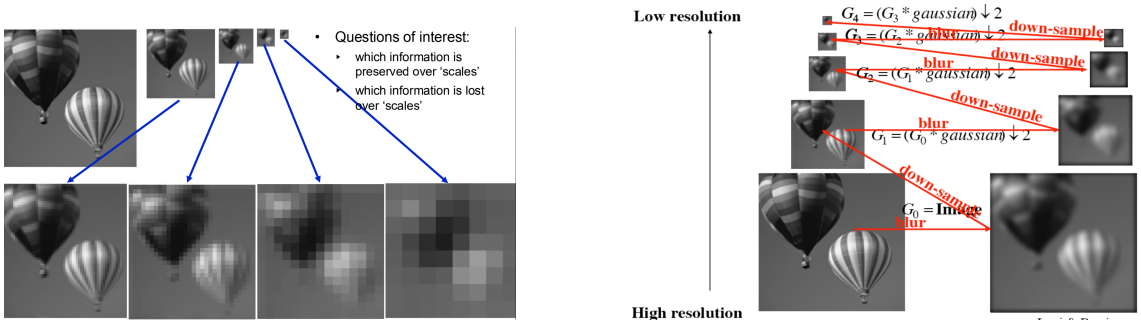


Figure 1.27: Gaussian Pyramid with blurring and downsampling at multiple scales.

1.8.2 Aliasing and the Need for Filtering

Aliasing

When high-frequency details in an image (such as sharp edges or fine patterns) are downsampled without first being blurred, aliasing occurs. Aliasing introduces distortions, such as: **Jagged lines, Incorrect textures, Strange patterns**. These distortions occur due to the undersampling of high- frequency details.

Gaussian Blur as a Solution

Before downsampling, applying a **Gaussian blur** acts as a **low-pass filter**, which removes or reduces high-frequency details:

- This avoids aliasing by smoothing sharp edges and fine details.
- The blurred image becomes more appropriate for representation at lower resolutions.

1.8.3 Digital Image Filtering

Recognition and Localization

Recognition involves identifying objects within an image, while **localization** determines where in the image these objects are located. To perform these tasks:

- **Filtering the image** to detect brightness changes (edges).
- **Fitting lines** to the detected edges to represent the object's contours.

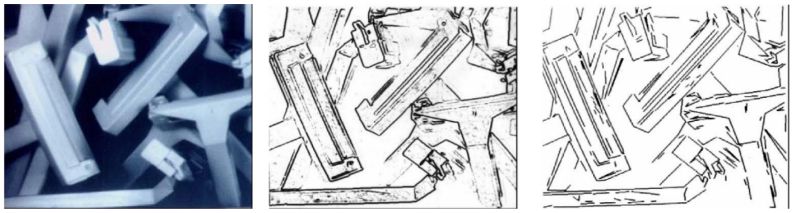


Figure 1.28: Filtering, fitting edges, and matching 3D models for object detection.

Another approach is **projecting a model** into the image and matching it to the detected edges, solving for the 3D pose of the object.

1.8.4 Gaussian Filtering

Gaussian filtering is a specific type of linear filtering where a Gaussian function is used to weigh nearby pixel values. This reduces noise and smooths the image.

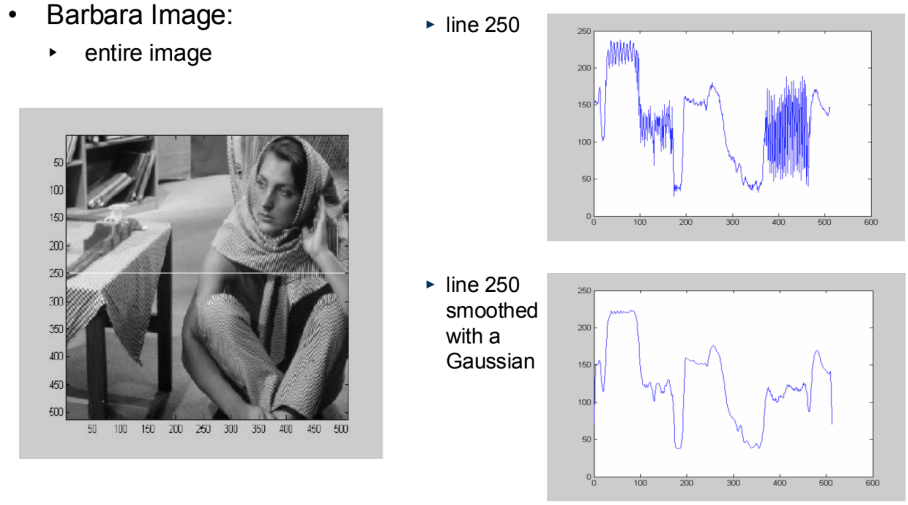


Figure 1.29: Example of Gaussian filtering applied to an image and its effects on smoothing and derivative calculation.

1.9 Edge Detection: Lesson 7/10

Edge detection involves identifying points in the image where there is a significant change in brightness. These points correspond to boundaries or edges between different regions in the image.

Ideal Edge Types

Edges can be classified into different types, including:

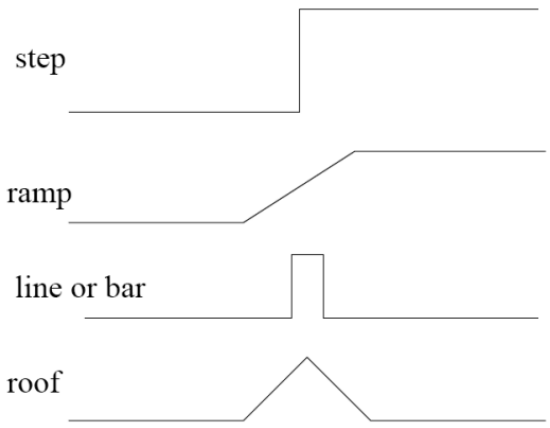


Figure 1.30: Different types of edges: step, ramp, line or bar, and roof.

1.9.1 Techniques for Edge Detection

There are two common techniques for edge detection:

- **First-order derivatives**: Measure the rate of change of pixel intensity values to detect edges.
- **Second-order derivatives**: Detect changes in the gradient of pixel intensities, often identifying the location of the edges more precisely.

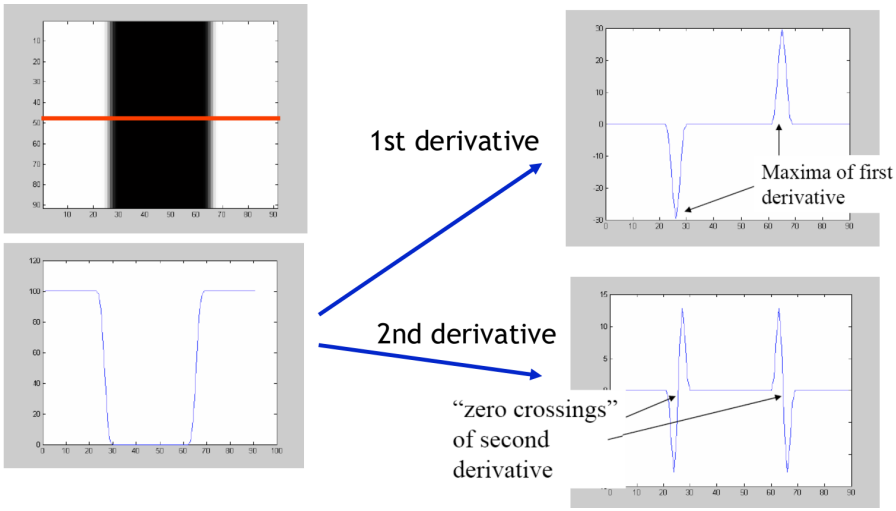


Figure 1.31: First and second derivatives of an image, showing maxima and zero crossings.

1.9.2 Mathematical Representation of Derivatives

The derivative of a function $f(x)$ can be represented as:

$$\frac{d}{dx}f(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \approx f(x+1) - f(x)$$

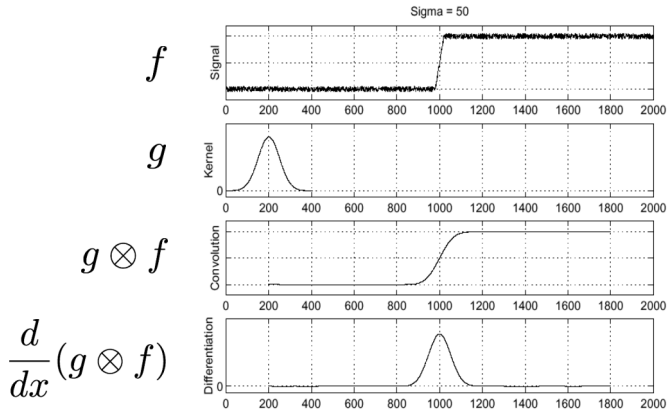


Figure 1.32: Derivative calculation and linear filter for edge detection.

To detect edges using the first derivative, follow this process:

- Smooth the image using a Gaussian filter.
- Calculate the derivative of the smoothed image.
- Find the maxima of the derivative to locate the edges.

We can simplify the computation since both the derivative and convolution are linear operations. This saves us one operation, as shown in the equation:

$$\frac{d}{dx}(g \otimes f) = \left(\frac{d}{dx}g\right) \otimes f$$

This property allows us to first compute the derivative of the Gaussian filter and then convolve it with the image, rather than computing the derivative after convolution.

1.9.3 Implementing 1D Edge Detection

- In 1D edge detection, we aim to identify significant changes (edges) in a signal by finding peaks in the first derivative that are "sufficiently large."
- The **hysteresis approach** uses two thresholds to robustly detect edges (High Threshold to identify strong peaks and Low Threshold to follow the edge curve). This allows for continuity across gaps where the gradient magnitude temporarily decreases.

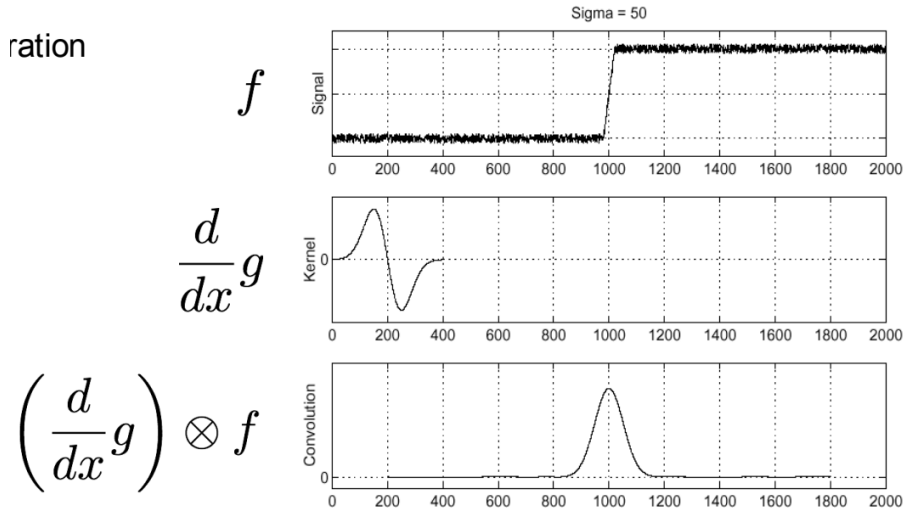


Figure 1.33: Illustration of the edge detection process using the Gaussian filter and derivatives.

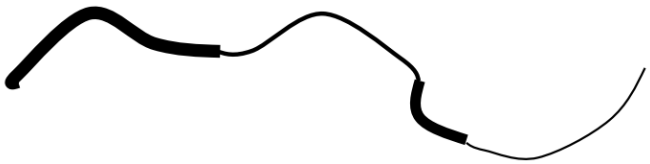


Figure 1.34: Hysteresis edge detection: Following edges through gaps.

1.9.4 Extension to 2D Edge Detection

Partial Derivatives

In 2D edge detection, we compute the partial derivatives of the image $I(x, y)$ with respect to x and y:

$$\frac{d}{dx}I(x,y) = I_x \approx I \otimes D_x \quad \text{and} \quad \frac{d}{dy}I(x,y) = I_y \approx I \otimes D_y$$

Where D_x and D_y are derivative kernels in the x- and y-directions, respectively.

Derivative Kernels

The kernels for the x- and y-direction derivatives are defined as:

$$D_x = \frac{1}{3}\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad D_y = \frac{1}{3}\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Derivative in the x-direction

We can simplify the derivative operation by applying the Gaussian filter first and then convolving with the derivative kernel:

$$D_x \otimes (G \otimes I) = (D_x \otimes G) \otimes I$$

This means that instead of first convolving the image with the Gaussian and then applying the derivative, we can compute the derivative of the Gaussian and convolve that with the image.

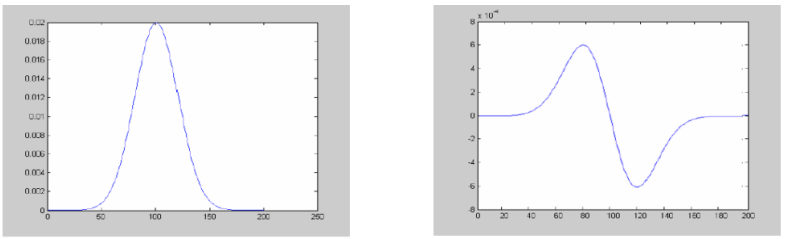


Figure 1.35: Derivative and smoothing process: 1D representations.

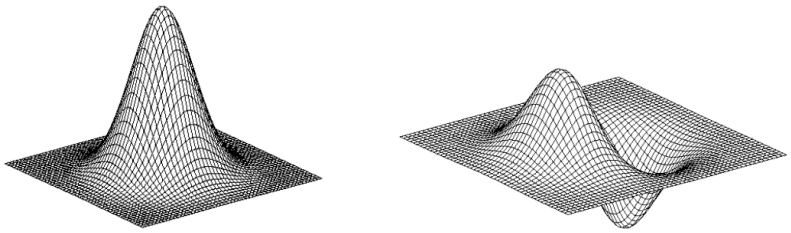


Figure 1.36: Derivative and smoothing process: 2D representations.

1.9.5 Gradient

The gradient of an image I is a vector that points in the direction of the most rapid intensity change. It is denoted as:

$$\nabla I = \begin{pmatrix} \frac{\partial I}{\partial x} & \frac{\partial I}{\partial y} \end{pmatrix}$$

The gradient helps detect changes in intensity along both the x-axis (horizontal) and the y-axis (vertical). For example:

- If there is a change along the x-axis, the gradient will be $(k, 0)$, meaning $\frac{\partial I}{\partial x} \neq 0$ and $\frac{\partial I}{\partial y} = 0$.
- If there is a change along the y-axis, the gradient will be $(0, k)$, meaning $\frac{\partial I}{\partial y} \neq 0$ and $\frac{\partial I}{\partial x} = 0$.
- If there is a change along both axes, the gradient will be (k_x, k_y), meaning changes in both directions.

The gradient vector direction is always perpendicular to the edge. The magnitude of the gradient vector measures the edge strength, with a larger gradient magnitude corresponding to a sharper or more prominent edge.

$$\nabla I = \begin{pmatrix} I_x \\ I_y \end{pmatrix} = \begin{pmatrix} \frac{\partial I}{\partial x} \\ \frac{\partial I}{\partial y} \end{pmatrix}$$

The magnitude of the gradient is:

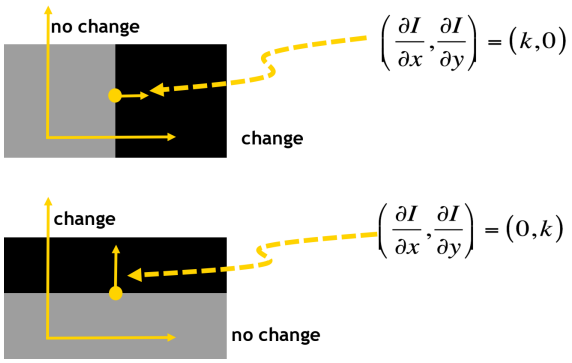


Figure 1.37: Identifying changes in intensity along different axes.

$$|\nabla I| = \sqrt{I_x^2 + I_y^2}$$

The direction of the gradient is:

$$\theta = \arctan\left(\frac{I_y}{I_x}\right)$$

1.9.6 Key Notes about Gradient

The behavior of the gradient magnitude depends on the scale of the smoothing filter:

- **Larger filters**: Smooth out more noise but can blur fine details, leading to thicker, more diffuse edges.
- **Smaller filters**: Preserve finer details but are more sensitive to noise, producing sharper but noisier edges.

Identifying Significant Points: To focus on the most important points in the gradient, we apply **Non-maximum suppression**. This technique finds local maxima in the gradient magnitude and suppresses values that are not the highest, thereby highlighting only the strongest edges.

Linking Relevant Points: Once the significant points are identified, the next step is linking them into continuous curves to represent the edges. This is achieved through:

- **Hysteresis thresholding**: Ensures that weak edges are connected only if they are part of a strong edge, maintaining edge continuity.
- **Edge tracking by gradient direction**: Traces edges by following the direction of the gradient, ensuring smooth edge detection even in noisy or broken regions.

1.9.7 The Canny Edge Detector

The Canny Edge Detector is a multi-step algorithm used to detect edges in images. It is widely regarded for its ability to detect true edges while minimizing noise. The steps involved are:

1. **Smoothing:** Apply a Gaussian filter to smooth the image and reduce noise.
2. **Gradient Calculation:** Compute the gradient magnitude and direction using the Sobel operator:
$$G = \sqrt{G_x^2 + G_y^2}$$
where G_x and G_y are the gradients in the x and y directions respectively.
3. **Non-Maximum Suppression:** Thins out the edges by suppressing pixels that are not local maxima along the gradient direction.
4. **Double Thresholding:** Apply a high and low threshold to classify pixels as strong, weak, or non-relevant edges.
5. **Edge Tracking by Hysteresis:** Connect weak edges to strong edges if they are connected to each other, forming continuous edges.

Non-Maximum Suppression In this step, pixels are checked to determine if they are local maxima along the gradient direction. If a pixel's gradient magnitude is greater than its neighbors (interpolated pixels p and r), it is preserved. Otherwise, it is suppressed.

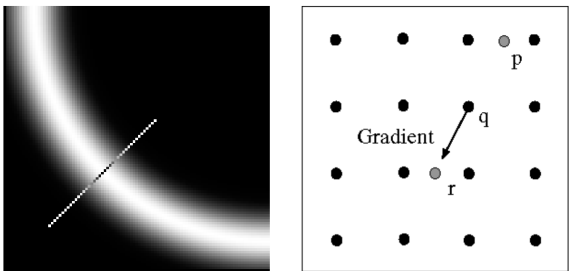


Figure 1.38: gradient magnitude along the edge and non-maximum suppression by checking interpolated pixels (p and r) along the gradient direction (q).

1.9.8 Edges and Derivatives

The detection of edges in images is based on the calculation of derivatives. The first derivative indicates the intensity change, while the second derivative helps locate zero-crossings, which are used to determine edges accurately.

- The **first derivative** identifies areas of rapid intensity change and is computed using gradient operators such as Sobel and Prewitt.

  The first derivative of a function $f(x)$ is defined as:

$$\frac{d}{dx}f(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

In discrete form, the first derivative can be approximated as:

$$\frac{d}{dx}f(x) \approx f(x+1) - f(x)$$

The mask used to compute the first derivative in digital images is:

$$\text{Mask for 1st derivative: } \begin{bmatrix} -1 & 1 \end{bmatrix}$$

- The **second derivative** provides information about the curvature of the function and is useful in pinpointing the exact location of edges, particularly through zero-crossings.

$$\frac{d^2}{dx^2}f(x) = \lim_{h\to 0} \frac{\frac{d}{dx}f(x+h) - \frac{d}{dx}f(x)}{h}$$

In discrete form, it can be approximated as:

$$\frac{d^2}{dx^2}f(x) \approx f(x+2) - 2f(x+1) + f(x)$$

The mask used for the second derivative is:

$$\text{Mask for 2nd derivative: } \begin{bmatrix} 1 & -2 & 1 \end{bmatrix}$$

1.10 The Laplacian

The Laplacian is a linear filter used in edge detection, but its direct computation is sensitive to noise.

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Therefore, it is often combined with Gaussian smoothing, forming the Laplacian of Gaussian (LoG). This method identifies edges as zero-crossings after smoothing the image.

$$\text{LoG}(x,y) = -\frac{1}{\pi\sigma^4}\left(1 - \frac{x^2+y^2}{2\sigma^2}\right)e^{-\frac{x^2+y^2}{2\sigma^2}}$$

The Laplacian can be approximated using the Difference of Gaussians (DoG) technique, which subtracts two Gaussian-blurred versions of the image at different scales:

$$\text{DoG}(x,y,\sigma_1,\sigma_2) = G(x,y,\sigma_1) - G(x,y,\sigma_2)$$

where σ_1 and σ_2 are the scales of the Gaussians. This method is efficient and reduces computational complexity.

1.10.1 The Laplacian Pyramid

The Laplacian Pyramid is a multi-scale representation technique used to detect edges at various scales. It applies the Laplacian operator at different levels of resolution, allowing for fine and coarse edge detection. The pyramid is built by:

1. Creating a Gaussian pyramid by progressively smoothing and downsampling the image.
2. Subtracting each level of the Gaussian pyramid from the next higher level to compute the Laplacian pyramid.

This method enables detection of features at multiple resolutions.

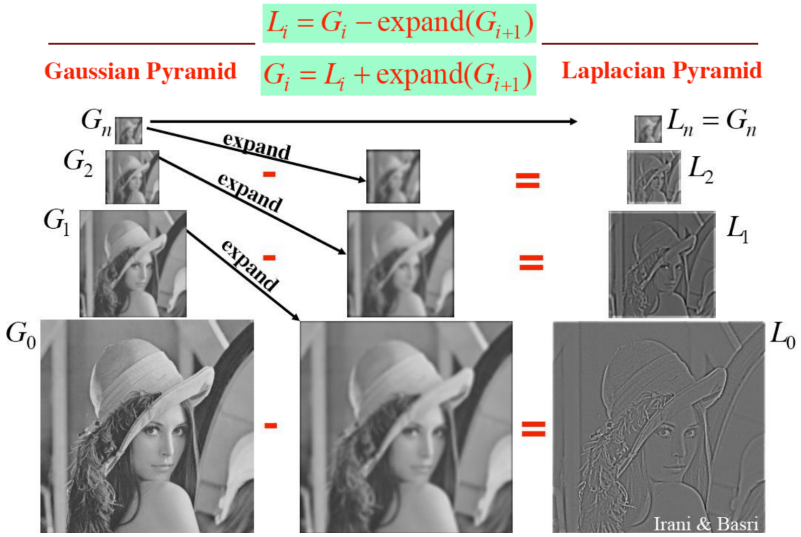


Figure 1.39: Laplacian pyramid

1.11 Object Recognition

Overview

Object recognition can be divided into two main types:

- **Object Identification:** Recognizing a specific instance of an object, such as your apple, your cup, or your dog. This is sometimes referred to as "instance recognition."
- **Object Classification:** Recognizing any instance of a category, such as any apple, any cup, or any dog. This is also called "generic object recognition" or "object categorization." It typically involves identifying objects at a 'basic level category.'

Challenges

When identifying objects, several challenges arise due to different modes of variation:

- **Viewpoint Changes:** Translation, image-plane rotation, scale changes, and out-of-plane rotation can affect recognition accuracy.
- **Illumination Variations:** Changes in lighting conditions.
- **Clutter and Occlusion:** Presence of other objects or partial obstruction of the object.
- **Noise:** Variations in image quality.

Appearance-Based Identification

The assumption in appearance-based identification is that objects can be represented by a collection of 2D images or "appearances." This approach doesn't require a 3D model and was a fundamental paradigm shift in the 1990s.

Global Representation for Object Recognition

Each view of an object is represented by a global descriptor. Recognition involves matching these global descriptors:

- **Invariance:** Descriptors can be made invariant to certain transformations like image-plane rotation and translation.
- **Training Data:** Variations such as viewpoint and scale changes can be incorporated during training.
- **Robustness:** Descriptors should be robust against noise, illumination changes, and partial occlusion.

1.11.1 Using Color for Recognition

Color remains constant under geometric transformations and can be used effectively for object identification:

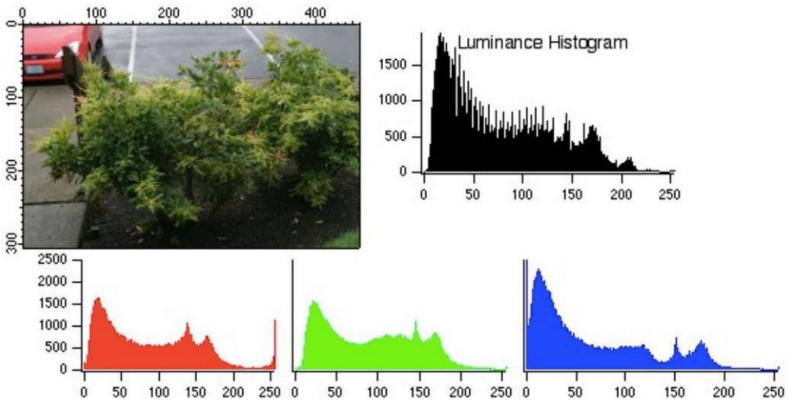
- **Local Feature:** Color is defined for each pixel, making it robust to partial occlusion.
- **Idea:** Directly using object colors for identification is effective, but using statistics of object colors (color histograms) provides better results.

Color Histograms

1D Color Histograms

A histogram counts the number of pixels with specific color values (R, G, and B) or luminance values. For example:

$$\text{Hist}(R) = \#(\text{pixels with color } R)$$

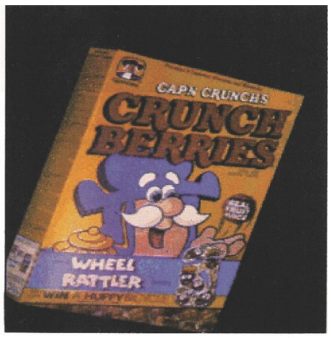


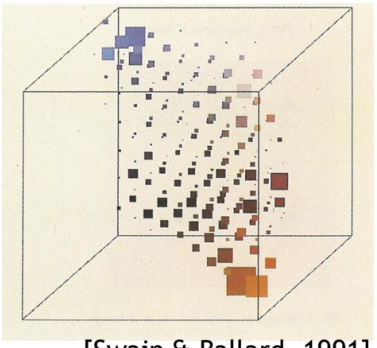
3D (Joint) Color Histograms

A 3D histogram is built based on the RGB values of each pixel:

$$H(R, G, B) = \#(\text{pixels with color } (R, G, B))$$

This allows for embedding the image in a space where closeness between colors has





[Swain & Ballard, 1991]

a meaningful interpretation.

Advantages of Color Histograms

Color histograms are robust in the presence of:

- Occlusion
- Rotation
- Partial visibility

They provide a robust representation of the object's appearance even under transformations.

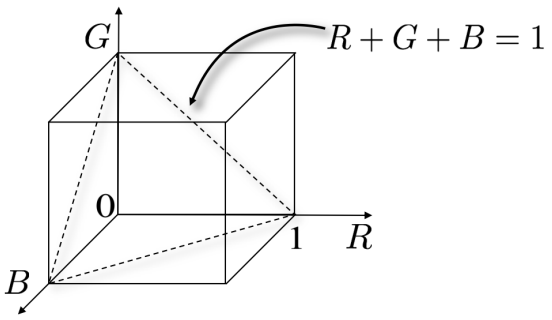
1.11.2 Chromatic Representation

Intensity and Normalization

One component of the RGB color space is intensity, given by:

$$I = R + G + B$$

Colors can be normalized based on intensity, allowing for a chromatic representation. Since $r+g+b=1$, only two parameters (e.g., r and g) are necessary, with b derived as $b=1-r-g$.

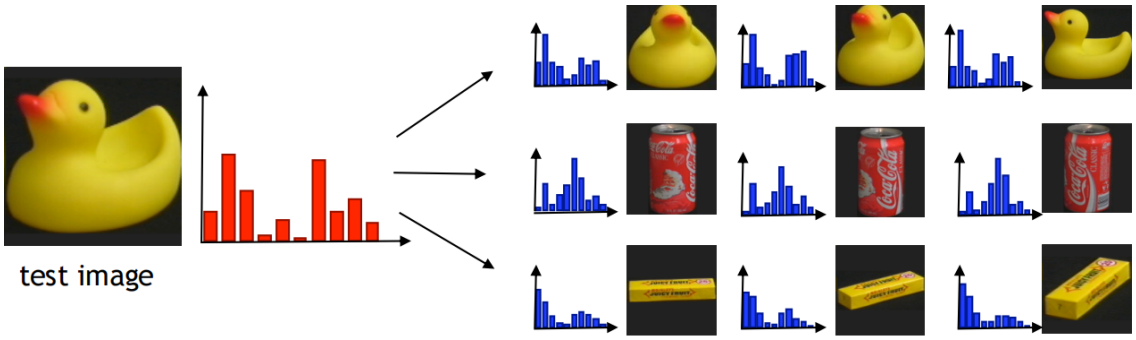


Recognition Using Histograms

To recognize objects using histograms:

1. Build a histogram database for each known.
2. Construct a histogram for the test image.
3. Compare the test histogram with each histogram in the database using a suitable comparison measure.

If the matching score is high enough, the object is identified; otherwise, the test image is rejected.



Comparison Measures

Several measures are used to compare histograms for object recognition:

- **Intersection:** This measures the overlap between two histograms Q and V, providing a value in the range [0, 1]. The formula is:

$$\cap(Q,V)=\sum_i \min(q_i,v_i)$$

For unnormalized histograms, the formula can be modified as:

$$\cap(Q,V)=\frac{1}{2}\left(\frac{\sum_i \min(q_i,v_i)}{\sum_i q_i}+\frac{\sum_i \min(q_i,v_i)}{\sum_i v_i}\right)$$

- **Euclidean Distance:** This focuses on the differences between the histograms. It considers each bin's difference and squares it, but is less discriminant. The range is [0, ∞]. The formula is:

$$d(Q, V) = \sum_i (q_i - v_i)^2$$

- **Chi-Square Distance:** This is a statistical measure that compares the distributions, often used to compute a significance score. It is more discriminant as it weights cells differently, but it can be sensitive to outliers. The formula is:

$$\chi^2(Q, V) = \sum_i \frac{(q_i - v_i)^2}{q_i + v_i}$$

Algorithm for Histogram-Based Recognition

A simple algorithm for object identification using histograms:

1. Build a set of histograms $H = \{M_1, M_2, M_3, \ldots\}$ for each known object.
2. Build a histogram T for the test image.
3. Compare T to each $M_k \in H$ using a comparison measure.
4. Select the object with the best matching score or reject if no object is similar enough.

This is referred to as the "Nearest-Neighbor" strategy.

PRO/CONS on Color Histograms

Advantages Color histograms offer several advantages:

- Invariant to object translations and image rotations.
- Slowly changing for out-of-plane rotations.
- Gradual change when parts of the object are occluded.
- Can recognize deformable objects, such as a sweater.

Limitations Despite their robustness, color histograms have limitations:

- Pixel colors change with illumination, a phenomenon known as the "color constancy problem."
- Changes in intensity and spectral composition (illumination color) can affect recognition.
- Not all objects can be identified based solely on their color distribution.

1.12 Performance Evaluation Metrics

Performance evaluation is crucial in comparing and validating the effectiveness of recognition algorithms. Different metrics and curves are used to measure the accuracy and reliability of models.

1.12.1 Single Value Metrics

Single value metrics provide a quick summary of model performance.

- **Accuracy:** Measures the proportion of correct predictions (both positive and negative) out of all predictions:

$$\text{Accuracy} = \frac{TP + TN}{N}$$

  where TP is True Positives, TN is True Negatives, and N is the total number of observations. This metric is straightforward but may not be ideal when the classes are imbalanced.

- **Precision:** Indicates the proportion of true positive predictions out of all positive predictions:

$$\text{Precision} = \frac{TP}{TP + FP}$$

  Precision is critical in applications where false positives are costly, such as medical diagnosis.

- **Recall (Sensitivity):** Reflects the proportion of actual positives that are correctly identified:

$$\text{Recall} = \frac{TP}{TP + FN}$$

  Recall is essential when minimizing false negatives is important, such as in disease detection.

- **F1-Score:** Combines precision and recall into a single metric using their harmonic mean:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

  The F1-Score is useful when we need a balance between precision and recall.

Precision and Recall Trade-off Precision and recall often present a trade-off:

- High precision indicates fewer false positives, but it may reduce recall.
- High recall means more true positives, but it might increase false positives.

To optimize these metrics, models often adjust thresholds

1.12.2 Confusion Matrix and Classification Outcomes

To better understand the relationships between predicted and actual outcomes, a confusion matrix is used. It shows the breakdown of:

	Predicted = 0	Predicted = 1
Actual = 0	True Negatives (TN)	False Positives (FP)
Actual = 1	False Negatives (FN)	True Positives (TP)

Figure 1.40: classification matrix

- **True Positives (TP):** Correctly predicted positive cases.
- **True Negatives (TN):** Correctly predicted negative cases.
- **False Positives (FP):** Incorrectly predicted positive cases (Type I error).
- **False Negatives (FN):** Incorrectly predicted negative cases (Type II error).

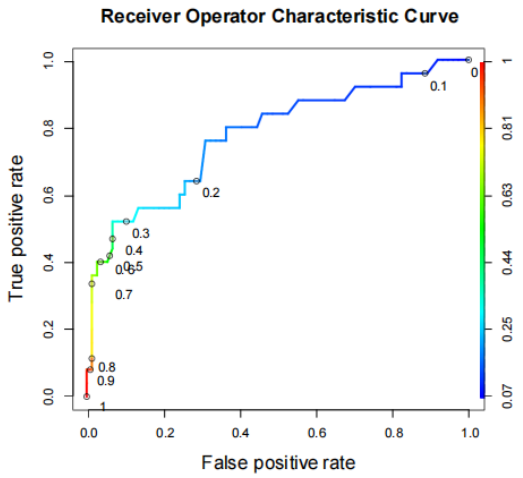
These values are the basis for calculating most performance metrics and help visualize where the model makes mistakes.

1.12.3 Receiver Operating Characteristic (ROC) Curve

The ROC curve is a plot of the True Positive Rate (TPR) against the False Positive Rate (FPR) as the threshold changes:

$$\text{True Positive Rate (TPR)} = \frac{TP}{TP + FN}$$

$$\text{False Positive Rate (FPR)} = \frac{FP}{FP + TN}$$



The area under the ROC curve (AUROC) indicates the model's ability to differentiate between classes:

- **AUROC = 1:** Perfect classification.
- **AUROC = 0.5:** Random guessing.

1.12.4 Threshold-Based Evaluation and Curves

The performance of classification algorithms often depends on the chosen threshold value t for classification. Choosing the optimal threshold t for a classifier involves balancing various metrics:

- A low threshold increases True Positive Rate (TPR) but may also increase False Positive Rate (FPR).
- A high threshold decreases FPR but may result in lower recall (more false negatives).

Using ROC and PR curves, we can visualize these trade-offs and choose the threshold that balances the desired metrics effectively.

Precision-Recall (PR) Curve

The PR curve plots precision against recall for different thresholds. It is particularly useful when dealing with imbalanced datasets, as it focuses on the positive class:

- High recall with high precision indicates excellent performance.
- The area under the Precision-Recall curve (AUPRC) gives a quantitative measure similar to AUROC.

1.12.5 Advanced Performance Metrics

While basic metrics provide valuable insights, advanced metrics like Log-Loss and Brier Score assess models' probabilistic outputs.

Log-Loss

Log-Loss measures the uncertainty of predictions by penalizing wrong predictions more if the model is confident:

$$\text{Log-Loss} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)]$$

where y_i is the actual label and p_i is the predicted probability. Lower Log-Loss values indicate better calibrated models.

Brier Score

The Brier Score quantifies the accuracy of probabilistic predictions. It is calculated as:

$$\text{Brier Score} = \frac{1}{N}\sum_{i=1}^{N}(p_i - y_i)^2$$

A lower Brier Score indicates better probability calibration.

Chapter 2

Introduction to Linear Regression

Linear regression is a supervised learning algorithm used for predicting real-valued outputs. It models the relationship between one or more input features and a target variable using a linear function.

2.0.1 Univariate Linear Regression

In the case of univariate linear regression, the model involves one feature x and predicts the output y using the hypothesis:

$$h_\theta(x) = \theta_0 + \theta_1 x \tag{2.1}$$

where θ_0 is the intercept and θ_1 is the slope of the line. The goal is to choose parameters θ_0 and θ_1 such that the hypothesis fits the training data well.

The training set consists of m samples (x_i, y_i), which are used to learn the parameters. The dataset can be represented as:

$$S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\} \tag{2.2}$$

The objective is to find a function $h(x)$ that approximates the target value y as closely as possible.

Assumptions Linear regression assumes that the joint probability distribution $p(X, Y)$ is the same for both the training and test datasets. The training examples are assumed to be identically distributed and independently distributed (i.i.d).

2.0.2 Regression Function

The regression function for univariate linear regression is given by:

$$f_\theta(x) = \theta_0 + \theta_1 x \tag{2.3}$$

where θ_0 and θ_1 are parameters to be learned from the training data.

For convenience, the input feature vector x_i can be represented as:

$$x_i = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \in \mathbb{R}^m, \quad y_i \in \mathbb{R} \tag{2.4}$$

The graph illustrates different linear functions with varying parameter values, for example:

- $\theta_0 = 0$, $\theta_1 = 1$: This represents a line passing through the origin with a slope of 1.
- $f(x) = 1.5$: A horizontal line at $y = 1.5$.

2.1 Training and Learning

1. **Training Set**: A set of training examples used to learn the model.
2. **Learning Algorithm**: Given input features such as house size and other characteristics, the learning algorithm estimates the target variable (e.g., the price of a house).

2.2 Multivariate Linear Regression

For multivariate linear regression, multiple features are used. The hypothesis function is:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n = \theta^T x \tag{2.5}$$

where x is the feature vector $[x_0, x_1, \ldots, x_n]^T$ with $x_0 = 1$ and θ is the parameter vector.

2.3 Cost Function

The cost function measures how well the hypothesis fits the training data. It is defined as:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 \tag{2.6}$$

where m is the number of training examples, $x^{(i)}$ is the i-th training example, and $y^{(i)}$ is the corresponding target value.

The cost function represents the Mean Squared Error (MSE) between the predicted values and the actual target values.

Note: Gradient descent might lead to a local minimum instead of the global minimum. Therefore, initialization is very important, and it is often recommended to set a seed for reproducibility.

2.4 Gradient Descent Algorithm

Gradient descent is an optimization algorithm used to minimize the cost function by iteratively updating the parameters θ in the direction of the steepest descent.

2.4.1 Gradient Descent Update Rule

The gradient descent update rule for linear regression is given by:

$$\theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j} \tag{2.7}$$

where α is the learning rate. The partial derivative is computed as:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)} \tag{2.8}$$

2.4.2 Batch vs Stochastic Gradient Descent

- **Batch Gradient Descent**: Uses all training examples to compute the gradient at each step, which can be slow for large datasets.
- **Stochastic Gradient Descent (SGD)**: Updates parameters using one training example at a time, leading to faster convergence but more noise in the updates.
- **Mini-Batch Gradient Descent**: A compromise between batch and SGD, where a small batch of examples is used to update the parameters.

2.5 Normal Equation

The normal equation provides an analytical solution to linear regression without requiring iteration. The parameters θ can be computed as:

$$\theta = (X^T X)^{-1} X^T y \tag{2.9}$$

This method is computationally expensive for large feature sets but avoids the need to choose a learning rate.

2.6 Feature Scaling and Mean Normalization

To improve the convergence of gradient descent, it is important to ensure that features are on a similar scale.

- **Feature Scaling**: Dividing each feature by its range to bring them into similar ranges.
- **Mean Normalization**: Subtracting the mean of each feature to ensure that the mean is approximately zero.

2.7 Locally Weighted Regression

Locally Weighted Regression (Loess/Lowess) is a non-parametric learning algorithm in which parameters grow with the data. The hypothesis is fitted to a local subset of data, providing flexibility for modeling complex datasets.

2.8 Probabilistic Interpretation of Least Squares

The least squares approach can also be interpreted probabilistically. Assuming the output y is generated by a linear function of the input plus Gaussian noise:

$$y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)} \tag{2.10}$$

where $\epsilon^{(i)}$ is normally distributed with mean zero, the likelihood function can be maximized to derive the least squares solution.

2.9 Correlation and Causation

- **Correlation**: Measures the linear relationship between two variables. The Pearson correlation coefficient r ranges between -1 and 1. When the values track each other well, the regression function fits the points better, indicating a stronger correlation.
- **Sum of Squared Residuals (SSR)**: A measure of the deviation of the predicted values from the actual values.
- **Pearson's Product Moment Correlation (PPMC)**: Measures the strength and direction of the linear relationship between variables. A value of 1 indicates maximum positive correlation, while a value of 0 indicates no correlation.
- **Coefficient of Determination (R^2)**: Represents the proportion of variance explained by the model. For simple linear regression, $R^2 = r^2$.
- **Causation**: Implies that changes in one variable directly cause changes in another.

2.10 Matrix Notation for Linear Regression

Linear regression can be expressed in matrix form, which makes computations more concise and allows for efficient implementation.

- X: Matrix of inputs of size $m \times n$.
- y: Vector of targets of size $m \times 1$.
- θ: Vector of parameters of size $n \times 1$.

The hypothesis can be expressed as:

$$h_\theta = X\theta \tag{2.11}$$

The cost function in matrix form is:

$$J(\theta) = \frac{1}{2m}\|y - X\theta\|^2 \tag{2.12}$$

Differentiating the cost function to find the gradient:

$$\frac{\partial J(\theta)}{\partial \theta} = -\frac{1}{m}X^T(y - X\theta) \tag{2.13}$$

Setting the gradient to zero yields the normal equation:

$$\theta = (X^T X)^{-1} X^T y \tag{2.14}$$

2.11 Polynomial Regression

Polynomial regression extends linear regression by using polynomial features:

$$f(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_m x^m \tag{2.15}$$

This allows for modeling more complex relationships. However, higher-degree polynomials increase the risk of overfitting.