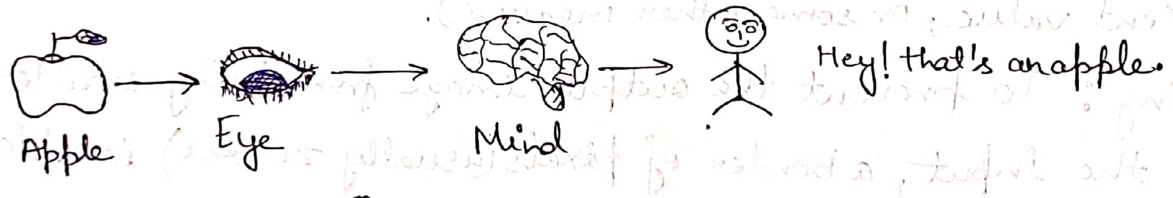
Introduction to Computer Vision

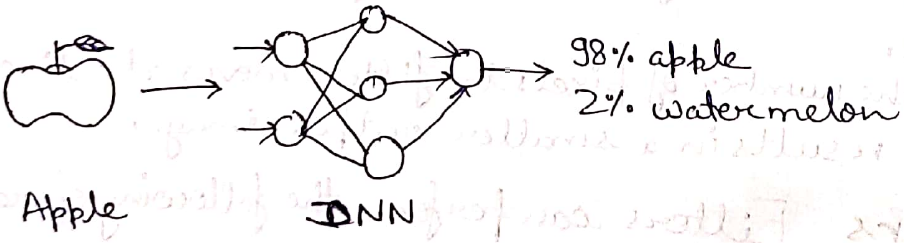
Computer Vision is a field of artificial intelligence that trains computers to interpret and understand the visual world.
CV enables machines to automatically extract, analyze and understand useful info from images/videos - emulating human visual perception.

Machine Vision Vs. Computer Vision

Machine vision is a practical application, often used in industrial settings. It involves both hardware (cameras, sensors) and software to perform specific tasks. It is used in inspecting products, guide robots.
Computer Vision is more about the algorithms and models that allows machine to understand and interpret images and videos. It is used in facial recognition, self-driving cars, medical image analysis, etc.

Human Vision vs. Computer Vision





Earlier computer vision systems relied on manually created rules and filters. These were traditional rule based CV systems.
Examples include techniques like Laplacian Smoothing and edge detection techniques like (Sobel, Canny, Laplacian, etc)
The traditional methods struggle with the vast number of variations and edge cases that can exist in real-world images.

The rise of deep learning with AlexNet

The AlexNet neural network, introduced in 2012, is highlighted as a major turning point for computer vision. Its key contribution include

→ it won the Imagenet challenge by large margin.
→ Usage of GPUs for faster training.
→ Introducing techniques like ReLU activation, Dropout regularization, Max Pooling and Data Augmentation

Classical Filters and Convolution : The heart of CV

Before deep learning exploded onto the scene, traditional CV was centered around filters. **Filters (Kernels)** are tiny matrices (usually 3x3 or 5x5) that are designed to detect specific features in an image, like edges or corners or texture. In traditional computer-vision, these filters were "hand-engineere meaning experts designed them on mathematical principles.

Convolution Operation : This is the process of sliding a filter (kernel) over an image, pixel by pixel to produce some transformation. At each position, we perform an element-wise dot product between the filter and the part of the image it's currently on. The sum of these products create a new pixel value in the output image often called as intensity (or gradient value, or some other measure).

→ Padding : To prevent the output image from being smaller than the input, a border of pixels (usually zeroes) is added around the image.

→ Stride : This is the number of pixels the filter moves at a time. A larger stride results in a smaller output image.

Purpose of Filters Filters can perform the following operations, making them useful in image processing —

(i) Feature Extraction → To detect edges, corners, textures. Finding out boundaries where there are sharp changes in brightness.

(ii) Noise Reduction → Smoothing out an image to reduce graininess. Filters like Gaussian blur can suppress pixel-level noise.

(iii) Image Enhancement → Some filters sharpen or enhance edges to make further analysis (like object segmentation) more robust.

(iv) Classical filters remain useful for preprocessing, computationally constrained systems, and for interpretable or real-time tasks where heavy neural networks aren't practical.

Edge detection techniques

The main idea behind edge-detection is to find where the **rate of change** of pixel intensity is high. In calculus, this is the **derivative**. Since images are discrete, we use filters to approximate these derivatives.

For an image whose pixel value can be described as $I(x,y)$, we can have partial derivatives $\partial I/\partial x$ (change in x-direction) and $\partial I/\partial y$ (change in y-direction).

(a) Sobel Filter :- This is a popular filter for approximating the first derivative of intensity. It's designed to be less sensitive to noise by giving more weight to the central pixels.

It has two versions — one for detecting vertical edges (changes in X-direction) and one for horizontal edges (changes in Y-direction).

The vertical filter (kernel) is given as — horizontal is also given below —

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \rightarrow \text{More weightage to central pixels} \qquad \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \rightarrow \text{Horizontal filter}$$

Using these additional pixels help to smoothenout impact of noise.

Why particularly this type of matrix?

The middle column (all zeroes) is for computing the difference in x-direction. If we sum left column's value that comes out to be -4 and right column's value comes out to be $+4$. This difference (left vs. right) approximates how big the change is along x-axis.

Notice the bigger weight in the middle row (-2 & 2). This is a "smoother" finite difference, giving extra weight to center row. It helps suppress noise that might exist at the top or bottom rows of 3×3 patch. When we convolve an image with this sobel kernel, we get an approximation of $\partial I/\partial x$.

(b) Prewitt Filter :- Similar to Sobel filter but simpler. It gives equal weight to all pixels in calculation. It also has both vertical and horizontal versions. While mathematically simpler, the Sobel filter is often preferred for its better noise reduction properties.

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \Rightarrow \text{x-direction (vertical)} \qquad \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \Rightarrow \text{y-direction (horizontal)}$$

(C) Laplacian Filter :- This filter approximates the second derivative of intensity. A second derivative sometimes makes edges appear more visually prominent. Unlike Sobel and Prewitt, the Laplacian filter detects all edges (vertical, horizontal, slanted) at once.

The second derivative amplifies rapid intensity changes more aggressively than the first derivative. It essentially looks for how fast the gradient itself is changing. This often produces sharper "peaks" at edges, making ~~ma~~ them appear more distinct in a second derivative map.

The Laplacian filter is given as—

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

The middle pixel has a large negative value (−4). This is equivalent to subtracting the "average" intensity around it. Neighbouring pixels each have smaller positive weight. This highlights places where there's a rapid change in intensity (the hallmark of edges).

Ques. Why do some filters like Sobel and Prewitt look symmetrical?

Ans→ (1) Balanced Response: Negative weights on one side, positive on the other side.

(2) No directional bias: The filter picks up changes in the same way if you move left to right or right to left.

(3) Less shifting: A symmetrical kernel won't shift features in the resulting image.